A Fast Scalable Implicit Solver for Nonlinear Time-Evolution Earthquake City Problem on Low-Ordered Unstructured Finite Elements with Artificial Intelligence and Transprecision Computing

Tsuyoshi Ichimura[1,2,3], Kohei Fujita[1,3], Takuma Yamaguchi[1], Akira Naruse[4], Jack C. Wells[5], Thomas C. Schulthess[6], Tjerk P. Straatsma[5], Christopher J. Zimmer[5], Maxime Martinasso[6], Kengo Nakajima[7,3], Muneo Hori[1,3], Lalith Maddegedara[1,3]

[1]Earthquake Research Institute & Department of Civil Engineering, The University of Tokyo
[2]Center for Advanced Intelligence Project, RIKEN, [3]Center for Computational Science, RIKEN
[4]NVIDIA Corporation, [5]Oak Ridge National Laboratory
[6]Swiss National Supercomputing Centre, [7]Information Technology Center, The University of Tokyo

***Abstract*—To address problems that occur due to earthquake in urban areas, we propose a method that utilizes artificial intelligence (AI) and transprecision computing to accelerate a nonlinear dynamic low-order unstructured finite-element solver. The AI is used to improve the convergence of iterative solver leading to 5.56-fold reduction in arithmetic count from a standard solver, and FP16-FP21-FP32-FP64 computing is used to accelerate the sparse matrix-vector product kernel, which demonstrated 71.4% peak FP64 performance on Summit. This is 25.3 times faster than a standard solver and 3.99 times faster than the state-of-the-art SC14 Gordon Bell Finalist solver. Furthermore, the proposed solver demonstrated high scalability (88.8% on the K computer and 89.5% on Piz Daint), leading to 14.7% peak FP64 performance on 4096 nodes of Summit. The proposed approach utilizing AI and FP16 arithmetic has implications for accelerating other implicit solvers used for earthquake city simulations as well as various fields.**

JUSTIFICATION FOR ACM GORDON BELL PRIZE

An implicit unstructured nonlinear dynamic low-order finite-element solver utilizing AI and FP16-FP21-FP32-FP64 computing obtained 19.8% peak FP64 performance on Piz Daint and 14.7% peak FP64 performance on Summit, leading to 25.3-fold speedup over a standard solver and 3.99-fold speedup over the SC14 Gordon Bell Finalist solver.

PERFORMANCE ATTRIBUTES

Category of achievement:
time-to-solution, peak performance, scalability
Type of method used: implicit
Results reported on the basis of:
whole application including I/O
Precision reported: mixed precision
System scale: results measured on full-scale system
Measurement mechanism:
timers, FLOP count (hardware counters)

I. OVERVIEW OF THE PROBLEM

Data analytics exemplified by machine-learning, which has high affinity with computer architecture and thus is straightforward to attain high performance, is overwhelming physics-based simulation and sweeping over the supercomputing field, even as if high peak performance is taken for granted. However, in the first place, reasonable development of both data analytics and physics-based simulation, and development of computation environment supporting these methods, is essential for extracting information from data. This important problem of integration of data analytics and physics-based simulation has been anticipated as a frontier in the supercomputing field. In this paper, we enhance the value of supercomputing by advancing this frontier from the point that the mathematical structure of data analytics and physics-based simulation have many parts in common. Our approach of combining data analytics and physics-based simulation based on the similar mathematical structure may be a turning point; its generalization is straightforward and is expected to bring large ripple effect to data analytics, physics-based simulation and the combination of both methods in the supercomputing field. We show the effectiveness of our approach by targeting the low-order unstructured finite-element method as it is the de facto standard in manufacturing industry and thus enlargement of problem size and acceleration of this method can lead to large benefits. Although the unstructured finite-element method is challenging to attain performance due to the random data access feature, we show our approach in attaining high performance on this selected problem.

By 2050, 70% of the global population is expected to live in cities; thus, cities are currently shifting toward *smart cities* capable of highly efficient urban activities, and supercomputing is expected to play a role in this shift. Smart cities are simulated based on three-dimensional city data, and large amounts of real-time observation data are analyzed using supercomputers to control city states effectively and optimize society's goals of productivity, safety, etc.. These analyses target both physical and virtual activities; thus, the scale of these simulations is expected to increase, making smart control and optimization of cities a frontier in supercomputing [1]. In addition, increasing urban density and efficiency leads to greater vulnerability; thus, resiliency should be improved significantly. Ignoring the need for resilient infrastructure could

SC18, November 11-16, 2018, Dallas, Texas, USA
978-1-5386-8384-2/18/$31.00

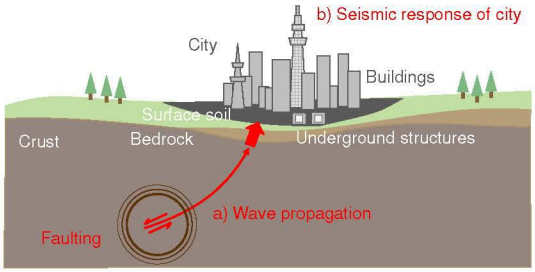

Fig. 1. The earthquake process consists of plate movement, wave generation, wave propagation, and seismic response of cities. Analyzing this process incurs significant computational costs because the target domain is large and the required resolution is fine. Physics-based simulation of cities, which is essential to realize smart cities, is particularly expensive, and is expected to be the focus of future SC Gordon Bell Sessions.

lead to a house of cards. Even the impression that such defects exist could damage a city's brand. Thus, improving resiliency is essential to realize smart cities. Natural disasters, such as earthquakes, can cause critical damage to cities. From this perspective, relative to realizing a smart city, we describe the computing innovations required to improve seismic resilience in cities.

Large earthquakes cause catastrophic damage to life, property, and society. Thus, they are considered a research area where supercomputing can play an important role [2]. Earthquakes are triggered by a fault caused by plate strain due to plate movement. Generated waves are propagated through the earth's crust (a in Fig. 1). Then the waves are amplified in soft soils near the surface, which cause shaking and result in fatalities and damage to property (b in Fig. 1). Reducing seismic damage is an important goal and has been an important topic at recent SC Gordon Bell Sessions, e.g., the SC08 Gordon Bell Finalist [3], SC14 Gordon Bell Finalist [4], and SC17 Gordon Bell Prize [5] targeted wave propagation, and the SC14 Gordon Bell Finalist [6] and SC15 Gordon Bell Finalist [7] targeted wave amplification and shaking. The latter papers focused on earthquake response of cities; however, these papers computed the ground response, and used these results to compute the response of buildings in a city. In other words, the coupling between the ground and urban structures was not considered. This is reasonable in terms of anti-seismic building design; however, relative to usability and resiliency in highly dense and optimized smart cities, fully coupled analysis of ground and urban structures is required. In this paper, we expand on the earthquake simulations targeted in past Gordon Bell Sessions and focus on fully coupled ground-urban structure seismic response analysis of cities. We employ computer-aided engineering (CAE) using three-dimensional urban data. We construct computer-based urban models, convert these models to numerical analysis models, and compute its response. The CAE approach is highly generalizable; thus, the insights gained here are expected to benefit CAE-based industries. The difference between standard CAE and this study is the scale of computation. Typical CAE is of the order of $10^{6\sim8}$ degrees of freedom; however, analysis models have a dimension greater than 10^{10} degrees of freedom when applied to cities. Thus, urban analysis becomes a frontier, and we make innovation for this challenging problem.

In this paper, we developed a method that utilizes artificial intelligence (AI) and transprecision computing to accelerate a CAE-based earthquake city simulation. The proposed approach has implications for accelerating other implicit solvers with AI and FP16 (half precision) arithmetic; thus, it is expected to have implications for earthquake city simulations as well as simulations in various fields.

II. Current State of the Art

Computing fully coupled ground and urban structure responses under large earthquake conditions involves nonlinear, time-history analysis of a large target domain with highly localized super high-resolution complex structures, i.e., the target domain is $10^3 \times 10^3 \times 10^{1\sim2}$ m with structures of complex geometry with feature size of $10^{-2\sim1}$ m. Complex geometry can be modeled with low compute cost with stress-free boundary conditions for nonlinear dynamic solid continuum equation using a three-dimensional finite-element method [8] with low-order solid elements (unstructured second-order tetrahedral elements) in double precision. The bottleneck in such analyses is the generation of finite-element models with dimension greater than 10^{10} degrees of freedom with complex geometry and performing $10^{3\sim4}$ time step nonlinear response analysis on these models. Modeling complex geometry with solid elements leads to locally small elements; thus, explicit time integration leads to very small time steps for stability, which, in turn, leads to significant computational costs. Here, we use implicit time integration (Newmark-β method (β = 1/4, δ = 1/2)) to avoid instability, although we have to solve huge matrix equations. The target nonlinear time history problem becomes

$$\left(\frac{4}{dt^2}\mathbf{M}+\frac{2}{dt}\mathbf{C}^n+\mathbf{K}^n\right)\delta\mathbf{u}^n = \mathbf{f}^n-\mathbf{q}^{n-1}+\mathbf{C}^n\mathbf{v}^{n-1}+\mathbf{M}\left(\mathbf{a}^{n-1}+\frac{4}{dt}\mathbf{v}^{n-1}\right), \quad (1)$$

where $\delta\mathbf{u}$, $\mathbf{u}$, $\mathbf{v}$, $\mathbf{a}$, and $\mathbf{f}$ denote incremental displacement, displacement, velocity, acceleration, and outer force vectors, respectively. $\mathbf{M}$, $\mathbf{C}$, and $\mathbf{K}$ denote the consistent mass matrix, damping matrix, and stiffness matrix, and dt and n denote the time step increment and time step number, respectively. We use Rayleigh damping for $\mathbf{C}$, where element damping matrix $\mathbf{C}_e^n$ can be written with element consistent mass matrix $\mathbf{M}_e$ and element stiffness matrix $\mathbf{K}_e^n$. The nonlinear time history problem is computed by solving Eq. (1) and by updating vectors using the obtained $\delta\mathbf{u}^n$ as $\mathbf{q}^n = \mathbf{q}^{n-1}+\mathbf{K}^n\delta\mathbf{u}^n$, $\mathbf{u}^n = \mathbf{u}^{n-1}+\delta\mathbf{u}^n$, $\mathbf{v}^n = -\mathbf{v}^{n-1}+\frac{2}{dt}\delta\mathbf{u}^n$, $\mathbf{a}^n = -\mathbf{a}^{n-1}-\frac{4}{dt}\mathbf{v}^{n-1}+\frac{4}{dt^2}\delta\mathbf{u}^n$.

Generation of super-complex urban models with more than 10^{10} degrees of freedom and computation of nonlinear time history problems with $10^{3\sim4}$ time steps are difficult problems. To the best of our knowledge, fully coupled ground-city nonlinear analysis has not been conducted to date. However, we have previously developed a method that can generate larger than 10^{10} degrees of freedom complex urban models [9] (Best Paper Award at HPC Asia 2018). As Eq. (1) dominates the compute cost of the analysis, the remaining difficulty toward nonlinear analysis of cities is to develop a fast and scalable solver suitable for this equation. The SC14 Gordon Bell Finalist *GAMERA* [6] uses a low-order unstructured finite-element method to conduct large-scale nonlinear seismic

analysis and is considered the state-of-the-art method for such problems. However, it only targets the ground; thus, *GAMERA* is designed to solve problems with significantly better characteristics (convergence of the system of equations) than the urban problem targeted in this study. Therefore, a new solver that is suitable for our target urban problem is required. Thus, we propose *MOTHRA*[1] in this paper. The standard method for solving such complex problems is *PCGE* (described in Section III), and we use *PCGE* for comparison. In summary, we compare the performance of the proposed *MOTHRA* solver to that of the state-of-the-art *GAMERA* and standard *PCGE* solvers. A previous study [7] solved large-scale nonlinear dynamic problems; however, that study took advantage of the uniformity of the target domain to speed up analyses. Unfortunately, exploiting domain uniformity is difficult with very complex problems, such as the target urban problem. Another study [9] computed the dynamic response of cities; however, that study only performed a linear analysis, i.e., nonlinearity was not considered. Relative to the limitations of these previous studies, we consider *GAMERA* to be the state-of-the-art solver.

III. Innovations Realized

Here, we summarize the design concept of *MOTHRA* with analyzing characteristics of the target problem, and then we explain how innovations are realized. Our target is to solve $\mathbf{A}^n\ \delta\mathbf{u}^n = \mathbf{b}^n$ in Eq. (1) derived from the three-dimensional nonlinear finite element method with unstructured second-order tetrahedral elements in double precision for each of the $10^{3\sim4}$ time steps with a relative error tolerance of 10^{-8}. Here, $\mathbf{A}^n = \left(\frac{4}{dt^2}\mathbf{M} + \frac{2}{dt}\mathbf{C}^n + \mathbf{K}^n\right)$ and $\mathbf{b}^n = \mathbf{f}^n - \mathbf{q}^{n-1} + \mathbf{C}^n\mathbf{v}^{n-1} + \mathbf{M}\left(\mathbf{a}^{n-1} + \frac{4}{dt}\mathbf{v}^{n-1}\right)$, and both change in each time step n. A characteristic of this problem is that the degrees of freedom of $\mathbf{u}^n$ are greater than 10^{10}. In addition, $\mathbf{A}$ is sparse; thus, access to $\mathbf{u}^n$ becomes random. As $\mathbf{M}$ is diagonally dominant, small dt leads to solvable characteristics of $\mathbf{A}^n$ even for very complex urban models. In addition, the problem size is large compared to the memory capacities of current computer systems. Therefore, it is impractical to allocate $\mathbf{A}^n$ in memory even when compressed formats such as compressed row storage are used. A well-known method for solving this problem is *PCGE* in double precision, that is a conjugate gradient (CG) solver, which uses a 3×3 block diagonal matrix of $\mathbf{A}^n$ as a preconditioner [10] (Fig. 2), and the element-by-element method (described later in Section III-B) which computes a matrix-vector product on the fly without storing $\mathbf{A}^n$ in memory [11]. However, as *PCGE* is computed in FP64 and ghost layers are updated by point-to-point communication in FP64, the computational performance of *PCGE* is insufficient on recent supercomputers with relatively fast arithmetic performance compared to memory and interconnect bandwidths. In addition, while *PCGE* does converge, its convergence is poor; thus, there is room for significant improvement relative to time-to-solution, which can be realized using a more sophisticated preconditioner. *GAMERA* can solve similar problems; however, it was designed for previous generation computers; thus, its performance is not optimal on recent supercomputers. In addition, our target problem

[1]iMplicit sOlver wiTH artificial intelligence and tRAnsprecision computing. Mother Guardian Kaiju; named after its form for searching solutions by changing shape of its system.

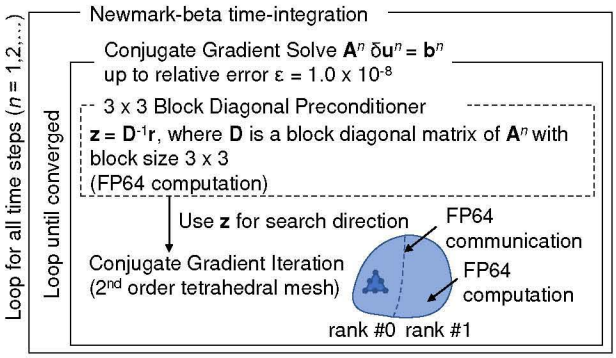

Fig. 2. Algorithm of *PCGE*. One matrix-vector product with FP64 computation and FP64 communication for updating ghost layers is conducted per CG iteration. Preconditioning is conducted using a 3×3 Block Diagonal matrix of $\mathbf{A}^n$. The light blue region denotes the target domain, while the dashed line denote the MPI partition boundary.

shows significantly poor convergence characteristics, which can be improved using more aggressive algorithms. We have developed a new solver *MOTHRA* enhanced by artificial intelligence (AI) and transprecision computing that is based on the idea of reducing data transfer by localization and homogenization in preconditioned CG solvers. This new solver obtains great performance on the largest current generation GPU-based Piz Daint and next generation GPU-based Summit supercomputers. Its significant performance is briefly summarized here (see Section V for details). The random access dominated low-order finite-element matrix-vector product core kernel of the implicit *MOTHRA* solver obtained 71.4% peak FP64 FLOPS on Summit's V100 graphic processing unit (GPU), and 19.5% peak FP64 performance for whole application on 288 GPUs of Summit. This is a 25.3-fold speedup over *PCGE* and a 3.99-fold speedup over *GAMERA*, representing 4.56-fold and 3.01-fold improvement in peak performance, respectively. The proposed solver attained high weak scalability (88.8%, 89.5%, and 75.5% efficiency on the K computer, Piz Daint, and Summit, respectively), leading to 14.7% peak FP64 performance on 4096 nodes of Summit. In addition, 93.4% strong scalability (576 to 2,304 processes) was achieved on the K computer. The following sections explains the details of major innovations with AI and transprecision computing and Fig. 3 summarizes the proposed method *MOTHRA*.

Although *MOTHRA* uses a sophisticated algorithm including AI, these are only used for the preconditioner. Thus, the numerical results of *MOTHRA* match with *PCGE* in adequate accuracy (relative error less than the error tolerance of 10^{-8}). We have checked the accuracy of *MOTHRA* on both performance measurement problems and the application problem, and confirmed that the same solution was obtained with higher computational performance.

A. Preconditioner enhanced by Artificial Intelligence

We use AI to improve the convergence of the solver. The poor convergence is due to the uneven distribution of the strength of graph connectivity (i.e., matrix component) in the target matrix. We train an AI (in particular, an Artificial Neural Network) to estimate the relationship between the localization of graph connectivity strength and the reduction

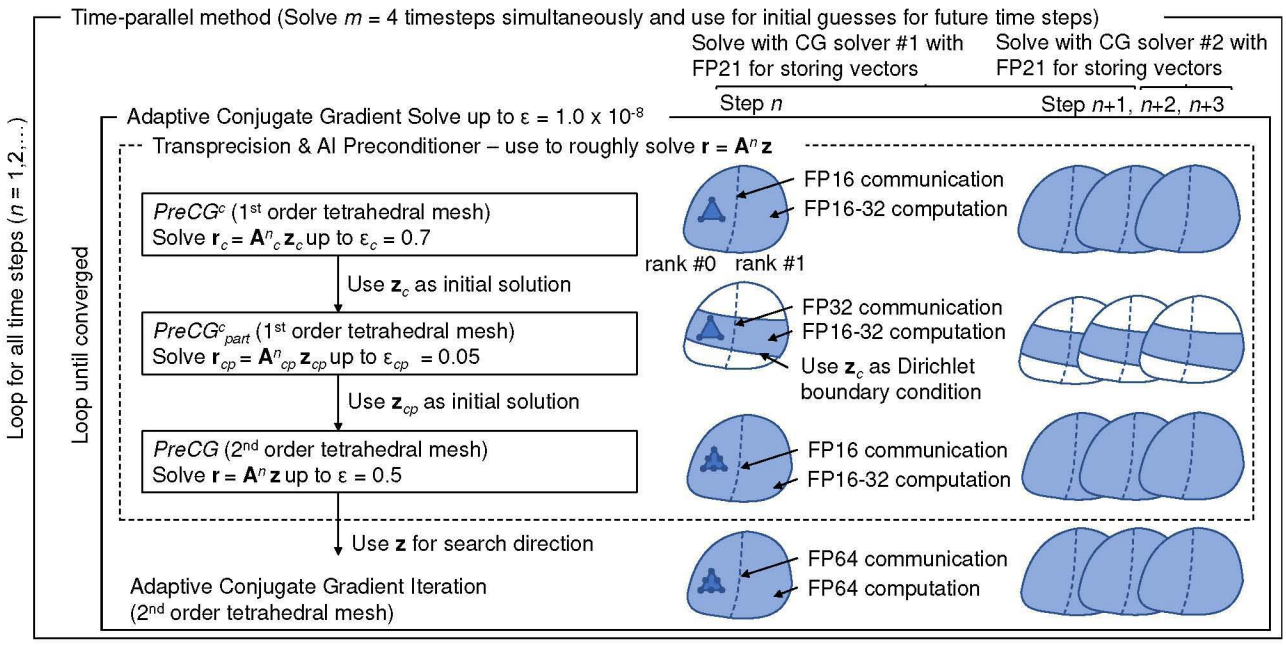

Fig. 3. Algorithm of implicit solver with AI and transprecision computing for nonlinear time history unstructured low-order finite-element simulations. The solver is based on the adaptive CG method, and the preconditioning equation $\mathbf{r} = \mathbf{A}^n\mathbf{z}$ is solved efficiently by the AI and transprecision computing. Here, preconditioning equation $\mathbf{r} = \mathbf{A}^n\mathbf{z}$ is first solved roughly by the linear tetrahedral element CG solver $PreCG^c$, and its solution is used as an initial solution for the linear tetrahedral element CG solver $PreCG^c_{part}$. Here, the light blue regions denote the target domain, while the dashed lines denote the MPI partition boundaries. The AI is used to estimate parts of the problem that have poor convergence characteristics. These parts are included in $PreCG^c_{part}$ part, which reduces computation cost compared to using $PreCG^c$. Next, the $PreCG^c_{part}$ results are used as an initial solution for the second-order tetrahedral element CG solver $PreCG$ to obtain highly accurate estimates for the preconditioning equation $\mathbf{r} = \mathbf{A}^n\mathbf{z}$. Transprecision is used in these preconditioners to reduce computational and communication costs (transprecision is used in EBE kernels as FP16-32, storing vectors in CG solvers as FP21, and in point-to-point communication as FP16 communication). As the search direction of the FP64 CG solver points to a very accurate solution, the number of iterations of the FP64 CG loops is reduced significantly; thus, nearly all computations can be conducted in low precision arithmetic. Furthermore, the time-parallel algorithm enables solving $m = 4$ time steps. We split the four time steps to group of two, and solve them in parallel. This leads to more concurrency in each CG solver, leading to overlap of point-to-point communication with memory bandwidth bound kernels. As FP64 is used in both computation and communication for all $PCGE$ components, we can expect significant speedup using $MOTHRA$ on recent computers with a large gap in arithmetic performance among high and low accuracy arithmetic.

of convergence, and we use this AI to estimate parts of the matrix that reduce convergence performance. Here, the idea is to reduce the concentration of graph connectivity strength as estimated by the AI with local operation to improve the solver's convergence. In other words, we train the AI using a data-set of graph connectivity strength and distribution of poor convergence in a small problem with similar graph characteristics as the larger target problem. We then use the AI to estimate parts of the larger target problem that result in poor convergence and perform a local operation to improve convergence. Note that the AI is only used to efficiently detect parts of the problem that result in poor convergence; therefore, degradation of the quality of the results does not occur, i.e., only the time to solution is improved. The graph connectivity dataset for a small-scale problem may be small in terms of supercomputing; however, it is a large and high-quality dataset when considered training data for the AI. Here, we stress the fact that small-scale datasets that are often disregarded in supercomputing can be utilized as large-scale high-quality datasets for AI training. Thus, these small-scale datasets have the potential to improve large-scale supercomputing.

This concept is straightforward and can be applied generally; however, developing an AI that can effectively estimate parts of the problem with poor convergence requires careful consideration. We explain our implementation in the following.

In preconditioned CG methods, preconditioning matrix $\mathbf{D}$ is used as $\mathbf{z} = \mathbf{D}^{-1}\mathbf{r}$ to improve convergence (Fig. 2). In contrast, in order to efficiently implement a more sophisticated preconditioner, we use the adaptive CG method, which solves $\mathbf{r} = \mathbf{A}^n\mathbf{z}$ roughly with another CG solver to improve convergence [12]. However, training an AI that estimates the relationship between graph connectivity strength and poor convergence in this form leads to poor generalization ability. As a result, we must select a suitable problem setting according to the characteristics of $\mathbf{A}^n$. We first homogenize this problem because second-order tetrahedral elements have two types of nodes (i.e., edge and vertex nodes) and the graph connectivity characteristics of these nodes differ completely. Rather than using the target second-order tetrahedral element problem $\mathbf{r} = \mathbf{A}^n\mathbf{z}$, we use a coarse but equivalent problem $\mathbf{r}_c = \mathbf{A}^n_c\mathbf{z}_c$ with linear tetrahedral elements. This leads to homogeneous graph connectivity characteristics for the target matrix $\mathbf{A}^n_c$. Here, we refer to solving $\mathbf{r} = \mathbf{A}^n\mathbf{z}$ using the CG method as $PreCG$, and we refer to solving $\mathbf{r}_c = \mathbf{A}^n_c\mathbf{z}_c$ with CG as $PreCG^c$ in the preconditioner. Next, we exploit the fact that the solution $\mathbf{u}$ can be expressed as $\mathbf{u}(\mathbf{x}) = \int \mathbf{G}(\mathbf{x},\mathbf{y})\mathbf{b}(\mathbf{y})dV$ using Green's function $\mathbf{G}$ and force distribution $\mathbf{b}$, which reflects the properties of the differential equation and model. Considering that the properties of $\mathbf{G}$ (its main part is given as a perturbation of $e^{|\mathbf{y}-\mathbf{x}|/c_2/dt}$ with time increment dt, shear wave velocity c_2 near point $\mathbf{x}$, and the distance d_{ij} between

TABLE I. PERFORMANCE OF AI. THE NUMBER OF ITERATIONS AND FLOP COUNT REQUIRED FOR SOLVING 25 TIME STEPS FOR MODEL W-1 DESCRIBED IN SECTION IV ON SUMMIT IS INDICATED.

	Without AI	With AI
CG iterations	132,665	88
$PreCG^c$ iterations	-	5,803
$PreCG^c_{part}$ iterations	-	26,826
$PreCG$ iterations	-	3,103
FLOP count	184.7 PFLOP	33.2 PFLOP

node i and its nearest node j), we expect that the connectivity strength of graph vertex i of $\mathbf{A}^n_c$ is governed with the frequency of strength of $d_{ij}/c_2/dt$. As $e^{|\mathbf{y}-\mathbf{x}|/c_2/dt}$ sharply attenuates with distance, the interaction between nodes are localized, and thus, the independence between parameters are enhanced, which leads to AI with higher performance. We trained our AI based on deep learning to relate this characteristic value to the residual error of a small-scale problem. Although it is very difficult to develop an AI with high accuracy on a heterogeneous problem, localization and homogenization to characterize the target problem leads to an AI with better generalization ability (SC17 Best Poster Award [13] is one example of constructing an AI with high generalization ability using such characteristics of the system). We use notation $\mathbf{A}^n_{cp}$ as part of $\mathbf{A}^n_c$ estimated as having poor convergence by AI, and we refer to solving $\mathbf{r}_{cp} = \mathbf{A}^n_{cp}\mathbf{z}_{cp}$ using CG as $PreCG^c_{part}$. Solving is preformed in the order $PreCG^c$, $PreCG^c_{part}$, and $PreCG$; thus, we expect improvement in convergence compared to when only $PreCG$ is used.

We use an AI trained by deep learning using a small part of the measurement problem described in Section IV as a training dataset, and use it to extract parts of $\mathbf{A}^n_c$ in the problem used in Section V-A. The resulting number of nodes in $\mathbf{A}^n_{cp}$ was 17,162,689, which corresponds to 10.9% of the total number of nodes in $\mathbf{A}^n_c$ (i.e., 157,339,343 nodes). 88 CG iterations, 5,803 $PreCG^c$ iterations, 26,826 $PreCG^c_{part}$ iterations, and 3,103 $PreCG$ iterations were required for solving 25 time steps with this problem. On the other hand, $PCGE$ required 132,665 CG iterations; thus, the convergence was improved by using the AI preconditioner (Table I). As the arithmetic count for each iteration of $PreCG^c$ and $PreCG^c_{part}$ is smaller than a CG iteration, this leads to reduction of arithmetic count by 184.7 PFLOP / 33.2 PFLOP = 5.56 times, which leads to significant speedup. As shown in Section V, applying the same AI to larger problems to extract $\mathbf{A}^n_{cp}$ leads to similar speedup; thus, we can see that the AI-based preconditioner is also extremely effective for larger problems.

B. Preconditioner enhanced by Transprecision Computing

As the preconditioning $\mathbf{r} = \mathbf{A}^n\mathbf{z}$ does not require high accuracy, we can use FP32 for $PreCG^c$, $PreCG^c_{part}$, and $PreCG$. We aim to improve computational efficiency by further use of the arithmetic space. Targeting flexible use of arithmetic space and improving system performance, support of a range of arithmetic data types is a trend in recent hardware. For example, IEEE754 FP16 [14] arithmetic units have been implemented in recent GPUs in addition to FP32 and FP64 arithmetic units [15], [16]. The trend of systems with relatively slow memory and interconnect bandwidth in comparison with fast arithmetic units is continuing; thus, reduction of data transfer sizes using low precision data types is also important for reducing time-to-solution. However, as the exponent and fraction of FP16 is limited (1 sign bit, 5 exponent bits, and 10 fractional bits), it is not straightforward to use FP16 for general scientific computations. Using FP16 in the preconditioning of implicit solvers is one idea; however, the range of values in a general matrix is large. Therefore, only very limited types of equations can be solved with meaningful accuracy when storing a global matrix in FP16. Indeed, the values of the matrix components have a large range for the target city problem; thus, direct use of FP16 is difficult, even for preconditioning purposes. On the other hand, from a finite-element method formulation perspective, the problem is discretized with local base functions in each element; thus, the range of values in each element is small. The large range in the components in the global matrix is caused by assembling elements with different characteristics. Thus, we use FP16 for element-wise computation, which is used for local expansion in the finite-element method. Here, we use FP16 for local matrix-vector products in the element-by-element (EBE) method, which is a matrix-free, matrix-vector product method. We describe the details of the transprecision EBE kernel in the following.

In the EBE method, matrix-vector product $\mathbf{y} \leftarrow \mathbf{A}\mathbf{x}$ is computed as follows:

$$\mathbf{y} \leftarrow \sum_e (\mathbf{Q}^{eT}(\mathbf{A}^e(\mathbf{Q}^e\mathbf{x}))). \tag{2}$$

Here, $\mathbf{A} = \sum \mathbf{Q}^{eT}\mathbf{A}^e\mathbf{Q}^e$, and $\mathbf{Q}^e$ is a mapping matrix between the local node number of element e and the global node number. As direct computation of Eq. (2) in FP16 leads to frequent overflow/underflow of variables, we use $\mathbf{x}^e_h \leftarrow \mathbf{f}(\mathbf{Q}^e\mathbf{x}_s)$, $\alpha^e_s \leftarrow g(\mathbf{A}^e_s)$, and $\beta^e_s \leftarrow h(\mathbf{Q}^e\mathbf{x}_s)$, and we compute $\mathbf{y}_s \leftarrow \sum_e \mathbf{Q}^{eT}\alpha^e_s\beta^e_s\mathbf{B}^e_h(\mathbf{x}^e_h)$. Here, subscripts s and h indicate values and functions in FP32 and FP16, respectively, and $\mathbf{f}$ and g represent functions that make the values of vector $\mathbf{x}^e_h$ and the values used in function $\mathbf{B}^e_h$ close to 1 (under infinite numerical accuracy, these functions satisfy $\mathbf{A}^e\mathbf{x}^e = g(\mathbf{A}^e)h(\mathbf{x}^e)\mathbf{B}^e(\mathbf{f}(\mathbf{x}^e))$). Although random addition to $\mathbf{y}_s$ and computation of scalars α^e_s, β^e_s are computed in FP32, the most costly $\mathbf{B}^e_h(\mathbf{x}^e_h)$ can be computed in FP16. Thus, use of this transprecision kernel in systems with high FP16 performance is expected to lead to shorter time-to-solution.

For EBE accelerated by FP16, the time involved in random load of $\mathbf{x}_s$ and random addition to $\mathbf{y}_s$ becomes dominant in the total elapsed time. To reduce random access time, we use a time-parallel algorithm [9]. Based on the idea that node-element connectivity is independent of time, random data access is reduced by solving several time steps in parallel. When m time steps are solved in parallel, the arithmetic count required for one iteration of the iterative solver increases m times compared to a standard solver. However, since the obtained solution for future time steps can be used for highly accurate initial solutions, we can reduce the total number of solver iterations by approximately $1/m$. Thus, using the time-parallel algorithm, we can reduce random access while maintaining the same arithmetic count, which leads to shorter time-to-solution on recent computers. In addition, by splitting the m time steps into two groups, we can run two independent solvers at the same time, which enables overlap of computation and communication among the two solvers. We have also implemented methods to utilize shared memory of GPUs to reduce atomic additions to L2 cache.

TABLE II. PERFORMANCE OF EBE MATRIX-VECTOR PRODUCT KERNEL (ELAPSED TIME PER VECTOR IS SHOWN)

System	Elapsed time	FLOPS efficiency to FP64 peak
One K computer node		
FP64 (one vector)	56.71 ms	10.9%
FP32 (one vector)	46.43 ms	13.2%
FP32 (two vectors)	18.53 ms	25.7%
One P100 GPU on Piz Daint		
FP64 (one vector)	785.3 us	19.1%
FP32 (one vector)	623.4 us	24.0%
FP16-32 (two vectors)	392.7 us	50.3%
One V100 GPU on Summit		
FP64 (one vector)	413.5 us	21.9%
FP32 (one vector)	400.3 us	22.8%
FP16-32 (two vectors)	178.2 us	71.4%

Table II shows the performance of the kernels. Here, FP64 (one vector) and FP32 (one vector) computes Eq. (2) directly in FP64/FP32 and are used with $PCGE$ and $GAMERA$, respectively. FP16-32 (two vectors) is the proposed transprecision EBE kernel with the time-parallel algorithm solved with $m = 4$ parallel time steps grouped into groups of two time steps. Note that elapsed time is normalized per vector. On Piz Daint and Summit with FP16 cores, FP16-32 (two vectors) improved FLOPS efficiency over FP64 by 2.63 and 3.26 times, respectively. As FP16 arithmetic is not supported on the K computer, i.e., the uniform precision of FP32 is used, we can see that the 2.35 times performance improvement is obtained by reducing data transfer sizes by changing from FP64 to FP32 and reducing the number of random accesses using the time-parallel algorithm.

Tensor Cores equipped on V100 GPUs, which are useful to attain high performance on data analytics, can also be used for physics-based simulations. For example, we have implemented a Tensor Core based algorithm for computing the mass matrix part of EBE kernel. Here, the EBE kernel is decomposed such that 16×16 matrix-matrix multiplication is frequently used. Although this modification required 1.21 times more FLOP count when compared to the baseline algorithm, the kernel was accelerated by 1.32-fold when compared with using FP32 cores on the V100 GPU. As Tensor Cores are extremely fast compared to the FP32 cores on V100 GPUs, redesigning the algorithm such that computation fits on Tensor Cores can be effective even if it requires more arithmetic counts. We can see that Tensor Core computing is not only effective for data analytics problems but also physics-based simulations by suitable algorithm development[2].

With GPU accelerated systems, the interconnect bandwidth and GPU device memory bandwidth is relatively slow compared to its high arithmetic performance per node; thus, reducing the amount of point-to-point communication and GPU device memory transfer is required to obtain scalability and performance. Thus, we use transprecision communication for the neighbor ghost layer updates of the EBE computation results, and use transprecision data storage for vectors stored in CG solvers. When vector data $\mathbf{y}_s$ are sent/received between two processes, we send vector $\mathbf{y}_h \leftarrow \mathbf{y}_s/|\mathbf{y}_s|$ and scalar $|\mathbf{y}_s|$, and we unpack the data by $\mathbf{y}'_s \leftarrow |\mathbf{y}_s|\mathbf{y}_h$ at the recipient (L infinity norm is used for $|.|$). Although $\mathbf{y}'_s$ does not match $\mathbf{y}_s$ exactly, the errors are localized at only MPI partition boundaries; thus, it does not change the convergence characteristics for most problems. The CG vector quantities are stored in a custom 21 bit data type with 1 sign bit, 8 exponent bits, and 12 fractional bits, which we call here FP21. We store three FP21 values in 64 bit memory, and convert them to three FP32 values for computation. After computation on FP32 cores, we convert them back to FP21 values and store them back into 64 bit memory. This adds computation for data conversion; however, since FP21 data types are only used for memory bandwidth bound kernels, reduction in data size by using FP21 leads to speedup even when this conversion is added. Furthermore, as the FP16 communication and FP21 data types are only used for preconditioning, it does not change the final results in double precision. As will be shown in Section V, the number of iterations when using FP16 communication and FP21 data types in CG vectors does not change the number of iterations significantly; thus, this leads to reduction in inter-node communication and device memory data transfer size by approximately 1/4 and 1/3, respectively, when compared to using FP64 data types.

[2]The implementation of the Tensor Core accelerated EBE kernel into the unstructured finite-element application is our future work; thus, Tensor Cores are not used in the measurements in Section V.

IV. HOW PERFORMANCE WAS MEASURED

A. System and Environment Where Performance was Measured

The $MOTHRA$ algorithm becomes faster on hardware with high transprecision kernel performance. In addition, we can expect improved scalability on large-scale systems where communication is a bottleneck. Thus, we measured performance on Piz Daint and Summit, which we also used to compare kernel performance. We also measured performance on the K computer to observe the differences in characteristics compared to $GAMERA$.

Summit [17]: Summit comprises 4,608 IBM Power System AC922 nodes, each with two IBM POWER9 (22 SIMD multi-core) processors and six NVIDIA Volta V100 accelerators. Each node has 512 GB of DDR4 memory for use by the POWER9 processors, and each of the six accelerators has 16 GB of 900 GB/s bandwidth high bandwidth memory (HBM2). The compute nodes are connected with a dual-rail EDR InfiniBand network with node injection bandwidth of 25 GB/s in both send/receive directions. Nodes are interconnected in a non-blocking fat tree topology, with SHArP technology designed to offload collective operation processing to the network [18]. The peak FP64 performance of a node is 7.8 TFLOPS × 6 = 46.8 TFLOPS, which yields a total of 215 PFLOPS for the entire system. The FP32 peak performance is double that of FP64, and FP16 peak performance is four times that of FP64 when using double-width arithmetic units. Tensor Cores with mixed FP16 and FP32 arithmetic are also available.

Piz Daint [19]: Piz Daint is a hybrid system comprising 1,431 Cray XC40 (CPU only) nodes and 5,320 Cray XC50 (CPU and accelerator) nodes. In this study, we used the XC50 nodes, each of which has an Intel Xeon E5-2690 v3 2.60 GHz CPU (12 cores) and an NVIDIA Tesla P100 accelerator. Each XC50 node has 64 GB of memory for use by the Xeon processors and 16 GB of 732 GB/s bandwidth HBM2 for

the accelerator. The compute nodes are connected by Cray Aries routing and communications ASIC with the Dragonfly network topology, which provides a node injection bandwidth of 10.2 GB/s in both the send/receive directions [20]. The peak FP64 performance of a node is 4.7 TFLOPS, which yields 25 PFLOPS for the entire system. The FP32 peak performance is double that of FP64, and FP16 peak performance is four times that of FP64 when using double-width arithmetic units.

K computer [21]: The K computer comprises 82,944 compute nodes, each with a single eight-core SPARC64 VIIIfx CPU. The FP64 peak performance of the CPU is 2 (GHz) $\times$ 2 (FMA) $\times$ 2 (SIMD width) $\times$ 2 (sets per core) $\times$ 8 (cores) = 128 GFLOPS, which yields 10.6 PFLOPS for the entire system. Note that the SIMD width does not change regardless of precision; thus, the peak performance for FP32 is the same. Each node has 16 GB of DDR3 SDRAM, with a peak bandwidth of 64 GB/s. Tofu, a six-dimensional interconnection network, is used for communication between nodes [22]. Each node can communicate simultaneously in four directions at 5 GB/s throughput in each direction.

B. What Application Was Used to Measure Performance

City problems that include underground structures are large in scale and becomes nonlinear during seismic shaking. In addition, the sharp coefficient contrast between materials leads to poor convergence characteristics. Thus, such problems are among the most costly in urban earthquake simulations. Therefore, accelerating the computation of these problems is expected to lead to speedup of other urban earthquake problems. Thus, we target a problem that mimics an underground structure with a hard layer in two-layered ground (Fig. 4). The hard layer is of linear material with properties similar to those of reinforced concrete underground structures, the first soil layer is a nonlinear material that softens due to strain, and the second soil layer is bedrock with linear material properties. To measure weak scaling, we duplicated this problem in the x, y directions. Although the problem settings are periodic, we used the METIS graph partitioning method [23] to partition the problem for parallel computation such that the load balancing characteristics were similar to those of an actual city problem. We input a wave with $dt = 0.01$ s (1995 JMA Kobe wave [24]) and measured the time required to solve 25 time steps. As the solution depends on dt and spatial discretization, we checked the numerical convergence of the results when fixing the problem setting and discretization sizes. We used semi-infinite absorbing boundary conditions on the sides and bottom of the model. Although we can use an arbitrary nonlinear constitutive model, we used the modified RO model [26] and the Masing rule [27] for nonlinear modeling of soils.

Using this problem, we compared the performance of the proposed $MOTHRA$, the standard $PCGE$, and the state-of-the-art $GAMERA$ solvers. As $GAMERA$ and $MOTHRA$ are both complex algorithmically, it is difficult to perform a rigorous comparison of component-wise costs. However, as both methods are based on adaptive CG methods, $GAMERA$ can be considered as $MOTHRA$ without $PreCG^c_{part}$ using AI, FP16/FP21 computation and communication, and the time-parallel algorithm. $PCGE$ can be considered as $GAMERA$ with FP64 computation and communication and disabling $PreCG^c$. Note that the relative error tolerance of all solvers

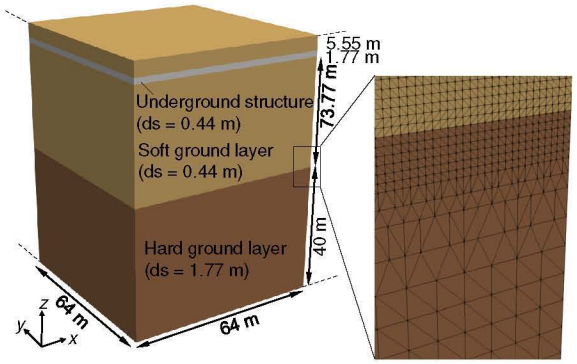

Fig. 4. Measurement model generated using [25]. The model is duplicated in the x and y directions to make a periodic problem suitable for measuring weak scaling.

was set to $\epsilon = 1.0 \times 10^{-8}$ for all problems. For $MOTHRA$, we used 0.7, 0.05, and 0.25 for error tolerances of the $PreCG^c$, $PreCG^c_{part}$, and $PreCG$ preconditioning solvers, respectively. For $GAMERA$, we used the error tolerances for the preconditioning solvers presented in the literature [6]. We used one GPU per MPI process for Piz Daint, one GPU per MPI process (6 MPI processes per node) for Summit and use eight OpenMP threads per MPI process for the K computer. We used MPI_Wtime to measure elapsed time on all systems. Hardware counters were used to measure FLOPS on the K computer, and FLOPS were measured on Piz Daint/Summit using nvprof. $PCGE$ was conducted in FP64, while $GAMERA$ was conducted in mixed precision of FP32 and FP64. We refer to the implementation of the transprecision computing algorithm following Fig. 3 as $MOTHRA$, while we refer to the algorithm in which FP16 communication is disabled (communication in FP16 in Fig. 3 was performed in FP32) as $MOTHRA$-a, which will be used to analyze communication performance. As the K computer does not support FP16 arithmetic and its memory bandwidth is relatively high compared to the floating point arithmetic performance, we conducted the computation and communication shown as FP16 and FP21 in Fig. 3 using FP32 (we refer to this implementation as $MOTHRA$-b). All measurements included outputting results to files.

V. Performance Results

In the city simulations targeted in this study, 10^{10} degrees-of-freedom nonlinear response problems must be solved for $10^{3\sim4}$ time steps on recent supercomputers within the $10^{0\sim1}$ h job time limit. As the memory sizes are small compared to the computational capability, we use full size of device memory to conduct analysis of a large scale problem. Thus, we target 12.3 million degrees-of-freedom per MPI process for Piz Daint and Summit for the measurement problem (Table III). As this problem size does not fit on each node of the K computer, we use 6.15 million degrees-of-freedom per MPI process on the K computer. The performance for problems with large degrees-of-freedom per MPI process is important for solving actual problems; thus, attaining weak scalability is important[3]. On the other hand, reducing the problem size per MPI process leads to

[3]As shown in Table III, the total number of iterations required for convergence with $PCGE$ was nearly constant among all models; thus, this model set is suitable for measuring weak scaling.

TABLE III. MODEL CONFIGURATIONS FOR PIZ DAINT/SUMMIT (WITHOUT BRACKETS) AND K COMPUTER (WITH BRACKETS). 1 GPU IS USED PER MPI PROCESS FOR SUMMIT/PIZ DAINT, WHILE 8 OPENMP THREADS ARE USED PER MPI PROCESS FOR K COMPUTER. 1 GPU IS USED FOR 2 K COMPUTER NODES. MODEL W-1 IS USED FOR STRONG SCALING MEASUREMENTS.

Model	# of MPI processes	Degrees-Of-Freedom	DOF per process	# of elements	*PCGE* iterations
W-1	288 (576)	3,545,198,451	12,309,716 (6,154,858)	883,104,768	132,665
W-2	576 (1,152)	7,088,615,271	12,306,623 (6,153,311)	1,766,209,536	131,320
W-3	1,152 (2,304)	14,174,736,543	12,304,458 (6,152,229)	3,532,419,072	129,765
W-4	2,304 (4,608)	28,345,910,535	12,302,912 (6,151,456)	7,064,838,144	128,165
W-5	4,608 (9,216)	56,687,546,151	12,301,984 (6,150,992)	9,419,784,192	126,475
W-6	6,144 (12,288)	75,580,545,159	12,301,521 (6,150,760)	18,839,568,384	125,636
W-7	12,288 (24,576)	151,152,541,191	12,300,825 (6,150,412)	37,679,136,768	123,910
W-8	24,576 (49,152)	302,293,683,783	12,300,361 (6,150,180)	75,358,273,536	-

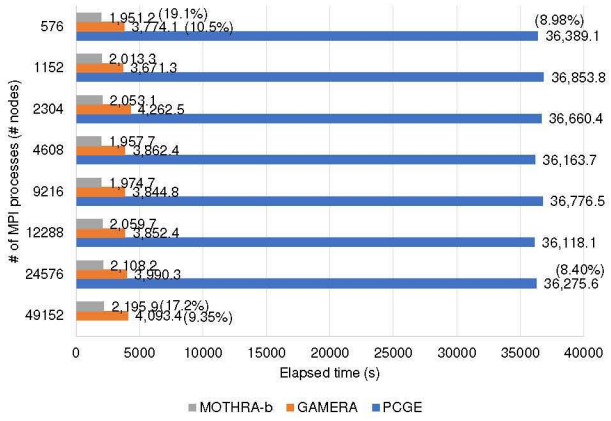

Fig. 5. Weak scaling results on K computer. Elapsed time and performance efficiency to FP64 peak are shown.

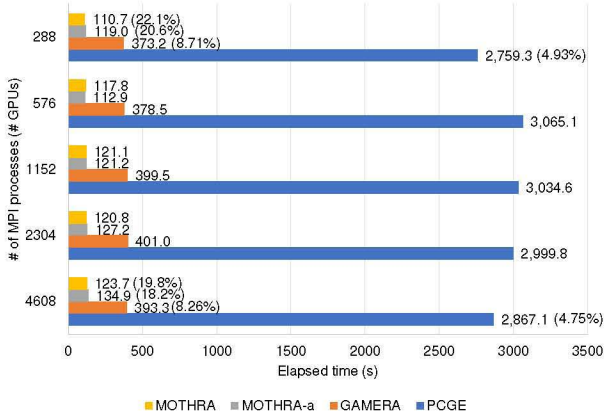

Fig. 6. Weak scaling results on Piz Daint. Elapsed time and performance efficiency to FP64 peak are shown.

reduction in solvable problem sizes; thus, small problem size per MPI process is a problem setting that will not be used for solving actual applications. Therefore, performance on such problem settings (i.e., performance for strong scaling at high node counts) is not important; thus, we should regard this as supplementary information.

A. Performance of MOTHRA Algorithm

We first compare performance of *PCGE*, *GAMERA* and *MOTHRA* using model W-1 in Table III. As shown in Fig. 5, 6 and 7, *MOTHRA* was 18.6-fold, 24.9-fold, and 25.3-fold faster than *PCGE* on the K computer, Piz Daint, and

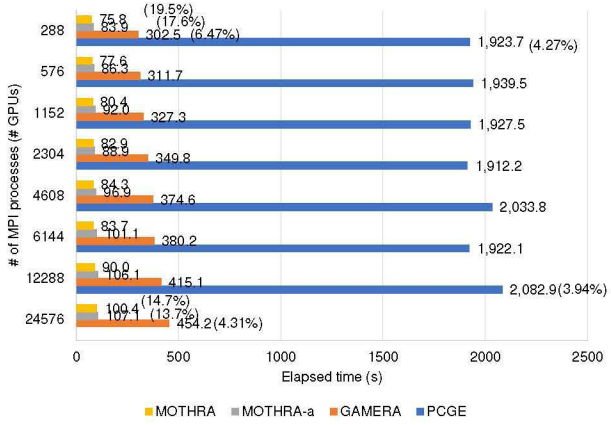

Fig. 7. Weak scaling results on Summit. Elapsed time and performance efficiency to FP64 peak are shown.

Summit, respectively. This is due to the reduced FLOP count and the improved peak performance; a 5.56-fold reduction of arithmetic count from *PCGE* (184.7 PFLOP) was obtained by *MOTHRA* (33.2 PFLOP) on Summit. Note that *GAMERA*, which also reduces arithmetic count by sophisticated preconditioning, still required 44.0 PFLOP, and this highlights the capability of *MOTHRA* to reduce arithmetic counts. Furthermore, the highly efficient transprecision EBE kernel of *MOTHRA* improved the *PCGE* FLOPS efficiency of 4.27% to 19.5% on Summit, leading to the 25.3-fold speedup. This is also 3.99-fold faster than the state-of-the-art *GAMERA*. The elapsed time ratio of Piz Daint to Summit was 110.7 s / 75.8 s = 1.46, which is close to the 1.66 difference in hardware peak FLOPS capability of the P100/V100 GPUs. Thus, the proposed solver can improve performance according to gains in hardware capabilities.

B. Scaling of Application on Large Scale Problem

First we see the weak scaling performance on the K computer (Fig. 5). Even though FP16 communication was disabled, we observed high scalability (88.8%) from W-1 (576 processes) to W-8 (49,512 processes) on the K computer. The high performance on a single node and high scalability leads to high peak performance of 17.2% of peak FP64 FLOPS with the largest problem (W-8), which yielded a 1.86-fold speedup over *GAMERA*. Next we see the effectiveness of FP16 communication on Piz Daint (Fig. 6). As the ratio of hardware FLOPS to interconnect bandwidth of Piz Daint is 4.7 TFLOPS : 10.2 GB/s, which is relatively slow compared to the 128 GFLOPS : 5 GB/s of the K computer, we can

expect improvement of performance and scalability using FP16 communication. The same solution was obtained with less than 5% difference in the number of iterations required for solving 25 time steps using FP32 communication ($MOTHRA$-a) and FP16 communication ($MOTHRA$), i.e., the iterations required for [CG, $PreCG^c$, $PreCG^c_{part}$, $PreCG$] was [119, 4516, 26593, 2448] and [117, 4453, 26553, 2422] for $MOTHRA$-a and $MOTHRA$, respectively. This leads to decrease of communication size by approximately 50%, which lead to 1.07 times speedup on W-1 (288 GPUs). The reduction of communication sizes lead to better scalability; the 88.2% weak scaling efficiency of $MOTHRA$-a was improved to 89.5% using $MOTHRA$ (measured from W-1 (288 nodes) to W-5 (4,608 nodes, near full system)). This lead to 23.2-fold speedup over $PCGE$ and 3.18-fold speedup over $GAMERA$ when measured on the full Piz Daint system. $MOTHRA$ attained 19.8% of FP64 peak on full Piz Daint, which is very high for a low-order unstructured implicit finite-element simulation on GPU supercomputers. Finally, we see weak scaling performance of $MOTHRA$ on Summit (Fig. 7). The FLOPS/network capability ratio of Summit is 46.8 TFLOPS : 25 GB/s, which is further reduced from Piz Daint; thus, $MOTHRA$ reducing communication is expected to be further effective. As expected, the performance of $MOTHRA$ using FP16 improved by 1.10 times from the version using FP32 communication ($MOTHRA$-a) on W-1. Together with high weak scalability of 75.5%, this lead to 4.52-fold speedup over $GAMERA$ on W-8 (4,096 nodes with 24576 GPUs, near full system). The performance of this run was 28.2 PFLOPS (14.7% of FP64 peak) on 4096 nodes of Summit.

This paper targets problems with large degrees-of-freedom per MPI process; thus, $MOTHRA$ is designed to solve such problems with high efficiency. Even under such circumstances, the strong scaling efficiency of $MOTHRA$ from 576 processes to 2,304 processes was 93.4% on the K computer, yielding speedup of 18.2-fold over $PCGE$ and 2.04-fold over $GAMERA$ with the greatest node count (Fig. 8). On Piz Daint and Summit, we obtained decent speedup relative to an increased number of processes leading to significantly shorter time-to-solution than the $GAMERA$ and $PCGE$ on the same node counts.

In summary, the proposed $MOTHRA$ solver obtained very high performance on small problems, i.e., 19.5% of FP64 peak on Summit and 22.1% of FP64 peak on Piz Daint. For the K computer with high interconnect capability, we attained high scalability without the use of FP16 communication. For Piz Daint and Summit, high scalability was attained by reducing point-to-point communication using the transprecision algorithm. These high scalability is due to the increased arithmetic counts per iteration with the time-parallel method, which results in reduced latency for synchronization between parallel computing elements in addition to the reduced communication.

VI. Implications

In this paper, we have shown that AI and transprecision can be used to improve the performance of iterative solvers by localization and selecting suitable basis functions for the target problem. Here, we used AI to train/estimate parts of the problem that show poor convergence characteristics to localize computation, and we used localization of finite

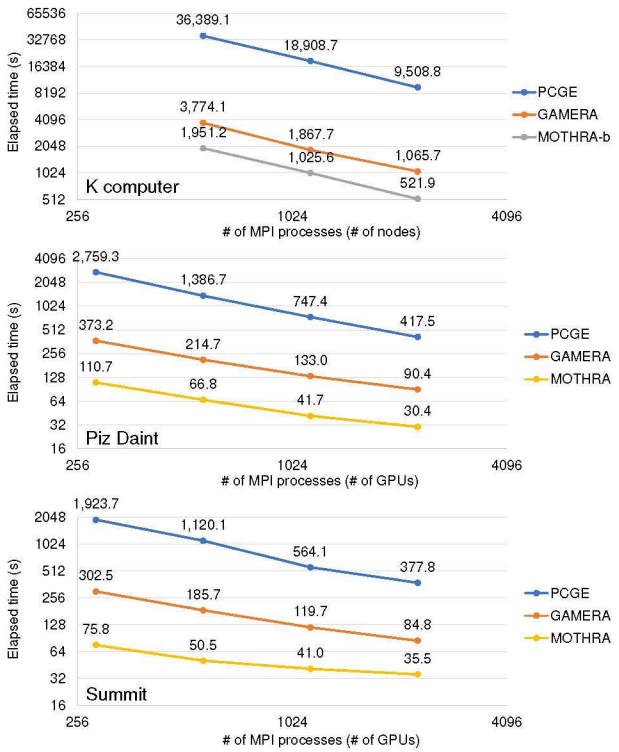

Fig. 8. Strong scaling results on the K computer, Piz Daint, and Summit

element basis functions to introduce transprecision computing. The random access dominated low-order finite-element matrix-vector product core kernel of the implicit $MOTHRA$ solver achieved 71.4% peak FP64 FLOPS on the Summit's V100 GPUs and achieved 19.5% peak performance for entire application on 48 Summit nodes. This is a 25.3-fold speedup over $PCGE$ and a 3.99-fold speedup over the state-of-the-art $GAMERA$ (improvement of 4.56-fold and 3.01-fold in peak performance, respectively). The proposed solver attained high weak scalability (88.8%, 89.5%, and 75.5% efficiency on the K computer, Piz Daint, and Summit, respectively), leading to 14.7% peak FP64 performance on 4096 nodes of Summit. In addition, 93.4% strong scalability (576 to 2,304 processes) was obtained on the K computer. These results demonstrate that AI and transprecision computing can be used for iterative solvers, and the fact that it can obtain high performance on recent supercomputers has implications for relevant applications. By using the proposed $MOTHRA$ solver on the most recent supercomputers, we can solve problems with high societal importance. For example, a seismic simulation of a large densely constructed city with complicated geometry comprising ground, underground structure, and high-rise buildings has been performed on Summit (Fig. 9; mesh generated using SGI UV300 and computed using 2,304 Summit GPUs, corresponding to problem size of W-4). As city problems involve large data uncertainty, and many stakeholders are involved in decision making and evaluation, realizing high quality computing in consideration of such uncertainties is becoming increasingly important. On the other hand, the nonlinearity of urban response is large; thus, computationally costly methods, such as Monte-Carlo simulations, are required to handle these uncertainties. Therefore, many cases of huge

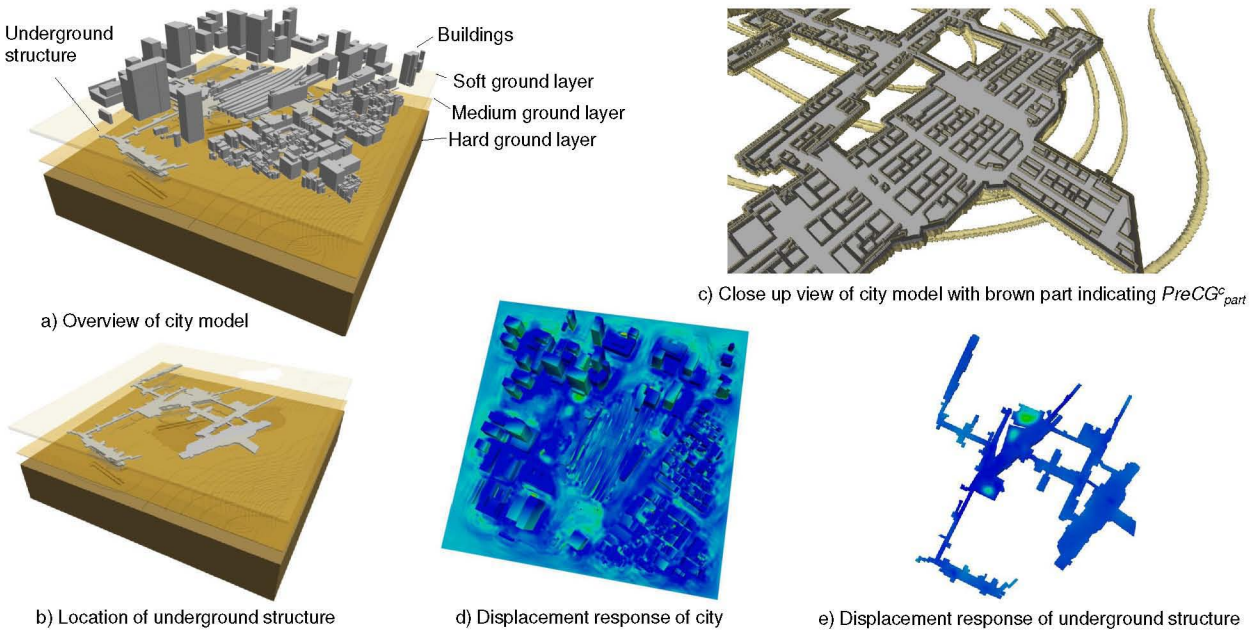

Fig. 9. Usage example of $MOTHRA$ computed on 384 Summit nodes (2,304 GPUs): a) overview of 1,024 m × 1,024 m × 370 m city model based on [28], [29] with underground and building structures surrounded by two-layered ground modeled with 3,961,851,160 elements and 16,291,917,564 degrees of freedom; b) location of underground structure; c) close-up of the underground structure modeled with 25 cm second-order tetrahedral elements (brown part indicates $PreCG^{c}_{part}$ part extracted by AI); d) displacement response to ground shaking; e) displacement response of underground structure

computations are required in city simulations. As a result, further accelerated solvers and faster computer systems are required. City simulation is and will continue to be a frontier.

To our knowledge AI and supercomputing has not been linked at such a scale before. The major difficulty of connecting AI with HPC is in the difference in accuracy of results: HPC results are accurate but expensive. On the other hand, although the accuracy is not as high, AI can solve problems that are impossible to be addressed by HPC. In our approach we redesign the HPC algorithm to take advantage of AI with less accuracy locally, accelerating the time-to-solution without loss of accuracy in the final results. We can understand that a new type of algorithm design can be enabled by use of AI. This approach of using AI within an HPC simulation is general and can be applied to other domains. We can expect more flexibility in computer system design with this increase of flexibility in algorithm design.

The fact that AI and transprecision computing can be used for HPC gives implications for future systems. We can expect use of low-precision arithmetic for accelerating other preconditioning algorithms; thus, development in hardware supporting low-precision arithmetic seems promising for improving performance per watt. Support in programming languages and compilers for FP16 data types are also anticipated to realize this on wide range of systems and applications. The use of AI in HPC supports the trend in equipping hardware specialized for conducting certain types of AI computation efficiently (e.g., Tensor Cores). On the other hand, the balance between arithmetic/memory bandwidth/interconnect bandwidth is expected to remain important in future system designs; although we circumvented the communication bandwidth bottleneck of Summit using FP16 communication, further reduction in relative interconnect performance will become difficult to achieve algorithmically. As hardware systems become more complicated, the importance of collaboration between computer and computational scientists/engineers is increasing, and codesign of systems and applications is required to realize the potential performance of future systems.

ACKNOWLEDGMENTS

Our results were obtained using the Summit at Oak Ridge Leadership Computing Facility, a US Department of Energy, Office of Science User Facility at Oak Ridge National Laboratory (ORNL), Piz Daint at Swiss National Supercomputing Centre (CSCS), and K computer at RIKEN Center for Computational Science (R-CCS, proposal numbers: hp170249, hp180217). We thank Yukihiko Hirano (NVIDIA) for coordination of the collaborative research project. We thank Christopher B. Fuson, Don E. Maxwell, Oscar Hernandez, Scott Atchley, Verónica Melesse-Vergara (ORNL), Jeff Larkin, Stephen Abbott (NVIDIA), Lixiang Luo (IBM), Richard Graham (Mellanox Technologies) for generous support concerning use of Summit. We thank Andreas Jocksch, Luca Marsella, Victor Holanda, Maria Grazia Giuffreda (CSCS) for generous support concerning use of Piz Daint. We thank the Operations and Computer Technologies Division of R-CCS and the High Performance Computing Infrastructure helpdesk for generous support concerning use of K computer. We thank Sachiko Hayashi of Cybernet Systems Co., Ltd. for support in visualizing the application example. We acknowledge support from Post K computer project [30] (Priority Issue 3 - Development of integrated simulation systems for hazards and disasters induced by earthquakes and tsunamis) and Japan Society for the Promotion of Science (18H05239, 26249066, 25220908, and 17K14719).

REFERENCES

[1] HPC Connects Plenary: The Century of the City, SC17: International Conference for High Performance Computing, Networking, Storage and Analysis, 2017.

[2] T.H. Jordan, SC15 Invited Talk Spotlight: Societal Impact of Earthquake Simulations at Extreme Scale, SC15: International Conference for High Performance Computing, Networking, Storage and Analysis, 2015.

[3] L. Carrington, D. Komatitsch, M. Laurenzano, M. Tikir, D. Michea, N.L. Goff, A. Snavely, and J. Tromp, "High-frequency simulations of global seismic wave propagation using SPECFEM3DGLOBE," Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'08), IEEE Press, 2008, pp. 1–11.

[4] A. Heinecke, A. Breuer, S. Rettenberger, M. Bader, A-A. Gabriel, C. Pelties, A. Bode, W. Barth, X-K. Liao, K. Vaidyanathan, M. Smelyanskiy, and P. Dubey, "Petascale High Order Dynamic Rupture Earthquake Simulations on Heterogeneous Supercomputers," Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'14), IEEE Press, 2014, pp. 3–14.

[5] H. Fu, C. He, B. Chen, Z. Yin, Z. Zhang, W. Zhang, T. Zhang, W. Xue, W. Liu, W. Yin, G. Yang, and X. Chen, "18.9-Pflops nonlinear earthquake simulation on sunway taihulight: Enabling depiction of 18-Hz and 8-meter scenarios," Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'17), ACM, 2017, pp. 2–13.

[6] T. Ichimura, K. Fujita, S. Tanaka, M. Hori, M. Lalith, Y. Shizawa, and H. Kobayashi, "Physics-based urban earthquake simulation enhanced by 10.7 BlnDOF x 30 K time-step unstructured FE non-linear seismic wave simulation," Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'14), IEEE Press, 2014, pp. 15–26.

[7] T. Ichimura, K. Fujita, P. E. B. Quinay, L. Maddegedara, M. Hori, S. Tanaka, Y. Shizawa, H. Kobayashi, and K. Minami, "Implicit Nonlinear Wave Simulation with 1.08T DOF and 0.270T Unstructured Finite Elements to Enhance Comprehensive Earthquake Simulation," Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'15), IEEE Press, 2015, pp. 1–12.

[8] O.C. Zienkiewicz and R. L. Taylor, The Finite Element Method for Solid and Structural Mechanics, Elsevier, 2005.

[9] K. Fujita, K. Katsushima, T. Ichimura, M. Horikoshi, K. Nakajima, M. Hori, and L. Maddegedara, "Wave Propagation Simulation of Complex Multi-Material Problems with Fast Low-Order Unstructured Finite-Element Meshing and Analysis," Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2018), ACM, 2018, pp. 24–35.

[10] Y. Saad, Iterative methods for sparse linear systems, SIAM, 2003.

[11] J. M. Winget and T. J. Hughes, "Solution algorithms for nonlinear transient heat conduction analysis employing element-by-element iterative strategies," Computer Methods in Applied Mechanics and Engineering, vol. 52(1–3), 1985, pp. 711–815.

[12] G. H. Golub and Q. Ye, "Inexact conjugate gradient method with inner-outer iteration," SIAM Journal on Scientific Computing, vol. 21(4), 1997, pp. 1305–1320.

[13] T. Ichimura, K. Fujita, T. Yamaguchi, M. Hori, M. Lalith, and N. Ueda, "AI with Super-Computed Data for Monte Carlo Earthquake Hazard Classification," Research Poster for SC17: International Conference for High Performance Computing, Networking, Storage and Analysis, 2017.

[14] D. Zuras, et al., "IEEE Standard for Floating-Point Arithmetic," IEEE Std 754-2008, 2008, pp. 1–70.

[15] NVIDIA Tesla P100 white paper, [Online]. https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf

[16] NVIDIA Tesla V100 GPU architecture, [Online]. https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf

[17] Summit, [Online]. https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/

[18] R. L. Graham, D. Bureddy, and P. Lui, "Scalable Hierarchical Aggregation Protocol (SHArP): A Hardware Architecture for Efficient Data Reduction," 2016 First International Workshop on Communication Optimizations in HPC (COMHPC), Salt Lake City, UT, 2016, pp. 1–10.

[19] Piz Daint, [Online]. https://www.cscs.ch/computers/piz-daint/

[20] B. Alverson, E. Froese, L. Kaplan, and D. Roweth, Cray XC Series Network, [Online]. https://www.cray.com/sites/default/files/resources/CrayXCNetwork.pdf

[21] H. Miyazaki, Y. Kusano, N. Shinjou, F. Shoji, M. Yokokawa, and T. Watanabe, "Overview of the K computer system," FUJITSU Sci. Tech. J., vol. 48(3), 2012, pp. 302–309.

[22] Y. Ajima, T. Inoue, S. Hiramoto, and T. Shimizu, "Tofu: interconnect for the K computer," FUJITSU Sci. Tech. J., vol. 48(3), 2012, pp. 280–285.

[23] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," SIAM Journal on scientific Computing, vol. 20(1), 1998, pp. 359–392.

[24] Strong ground motion of The Southern Hyogo prefecture earthquake in 1995 observed at Kobe JMA observatory, Japan Meteorological Agency, [Online]. https://www.data.jma.go.jp/svd/eqev/data/kyoshin/jishin/hyogo_nanbu/dat/H1171931.csv

[25] T. Ichimura, M. Hori, and J. Bielak, "A Hybrid multiresolution meshing technique for finite element three-dimensional earthquake ground motion modeling in basins including topography," Geophysical Journal International, vol. 177, 2009, pp. 1221–1232.

[26] I. M. Idriss, R. Dobry, and R. D. Sing, "Nonlinear Behavior of Soft Clays during Cyclic Loading," Journal of the Geotechnical Engineering Division, vol. 104(ASCE 14265), 1978, pp. 1427–1447.

[27] G. Masing, "Eigenspannungen und Verfestigung beim Messing," Proceedings of the 2nd International Congress of Applied Mechanics, 1926, pp. 332–335.

[28] 5m mesh digital elevation map, Tokyo ward area, Geospatial Information Authority of Japan, [Online]. https://www.gsi.go.jp/MAP/CD-ROM/dem5m/index.htm

[29] National Digital Soil Map, The Japanese Geotechincal Society, [Online]. https://www.denshi-jiban.jp/

[30] FLAGSHIP 2020 Project, Post-K Supercomputer project [Online]. https://www.r-ccs.riken.jp/fs2020p/en/