Development of Element-by-Element Kernel Algorithms in Unstructured Implicit Low-Order Finite-Element Earthquake Simulation for Many-Core Wide-SIMD CPUs

Kohei Fujita[1,2(✉)], Masashi Horikoshi[3], Tsuyoshi Ichimura[1,2,4], Larry Meadows[5], Kengo Nakajima[2,6], Muneo Hori[7], and Lalith Maddegedara[1,2]

[1] Earthquake Research Institute and Department of Civil Engineering, The University of Tokyo, Tokyo, Japan
{fujita,ichimura,lalith}@eri.u-tokyo.ac.jp
[2] Center for Computational Science, RIKEN, Kobe, Japan
[3] Core and Visual Computing Group, Intel K.K, Tokyo, Japan
masashi.horikoshi@intel.com
[4] Center for Advanced Intelligence Project, RIKEN, Tokyo, Japan
[5] Data Center Group, Intel Corporation, Hillsboro, USA
lawrence.f.meadows@intel.com
[6] Information Technology Center, The University of Tokyo, Tokyo, Japan
nakajima@cc.u-tokyo.ac.jp
[7] Japan Agency for Marine-Earth Science and Technology, Yokosuka, Japan
horimune@jamstec.go.jp

Abstract. Acceleration of the Element-by-Element (EBE) kernel in matrix-vector products is essential for high-performance in unstructured implicit finite-element applications. However, the EBE kernel is not straightforward to attain high performance due to random data access with data recurrence. In this paper, we develop methods to circumvent these data races for high performance on many-core CPU architectures with wide SIMD units. The developed EBE kernel attains 16.3% and 20.9% of FP32 peak on Intel Xeon Phi Knights Landing based Oakforest-PACS and Intel Skylake Xeon Gold processor based system, respectively. This leads to 2.88-fold speedup over the baseline kernel and 2.03-fold speedup of the whole finite-element application on Oakforest-PACS. An example of urban earthquake simulation using the developed finite-element application is shown.

Keywords: Finite-element method · Random data access · Many-core · SIMD

1 Introduction

Energy efficiency is one of the requirements for exascale computing, and many-core CPUs with wide SIMD units are considered to be one of the architectures

J. M. F. Rodrigues et al. (Eds.): ICCS 2019, LNCS 11536, pp. 267–280, 2019.
https://doi.org/10.1007/978-3-030-22734-0_20

towards this goal. For example, the Post-K supercomputer that is scheduled for operation around 2021 is announced to equip many-core CPUs with 512 bit wide SIMD cores [1]. However, some applications are difficult to benefit from wide SIMD and many-core architectures due to random data access and data recurrence, and thus algorithms enabling efficient use of these architectures are important towards exascale computing. The unstructured low-order implicit finite-element method is one of the applications that are not straightforward for efficient use of many-core wide SIMD CPUs. As the unstructured low-order implicit finite-element method is the de facto standard in structural simulations used in the manufacturing industry, the acceleration of this method is expected to lead to large ripple effects. In this paper, we focus on the unstructured low-order implicit finite-element method as a target for scalable algorithm development for many-core wide SIMD CPUs.

Many studies target implicit solvers on large scale systems. For example, the SC15 Gordon Bell Prize Paper [2] uses multi-grid solvers for simulation of extremely large mantle convection problems, and the SC16 Gordon Bell Prize Paper uses the Sunway Taihu Light Supercomputer to solve climate problems [3]. Also, an SC15 Gordon Bell Prize Finalist [4] solves an urban earthquake problem on the K computer. However, all of these simulations use structured or semi-structured mesh, which are both SIMD and multi-core friendly, for efficient computation of matrix-vector products for the implicit solvers. However, for solving general manufacturing or geoscience problems, use of pure unstructured mesh is required. The state-of-the-art of low-order unstructured finite-element method on CPU based systems is SC14 Gordon Bell Prize Finalist *GAMERA* [5]. Targeting architectures with high arithmetic capabilities and relatively low memory bandwidth, *GAMERA* uses a matrix-free kernel for computation of matrix vector products (i.e., the Element-by-Element kernel, EBE [6]). Here, instead of storing and reading the global matrix from memory, element matrices are generated and multiplied with the right hand side vector every time a matrix-vector product is called. This enables fast computation of matrix-vector products with good load balance on massively parallel systems with relatively low memory bandwidth, which lead to high application performance and high scalability. However, as *GAMERA* is targeted for the K computer which comprises 8 core CPUs with 2 wide SIMD cores [7], the computation algorithm used for the EBE kernel does not consider the wider SIMD and larger core counts of current and near future many-core CPUs. Thus, a new time-parallel computation algorithm *GHYDRA* was developed in a 2018 IPDPS paper [8], that enables use of uniformity of mesh in time domain to reduce random data access. This algorithm is suitable for recent wide SIMD CPUs; however, EBE kernel algorithms suitable for this type of architecture is required to fully exploit its performance. In this paper, we develop EBE kernel algorithms for efficient computation of the *GHYDRA* finite-element solver algorithm on many-core CPU systems, and compare with conventional EBE algorithms on the K computer, Intel Xeon Phi Knights Landing based Oakforest-PACS, and an Intel Skylake Xeon Gold CPU based system. We also show an urban earthquake simulation example using

the unstructured finite-element solver accelerated by the developed EBE kernel algorithms.

2 Finite-Element Solver Algorithm

2.1 Target Problem

Earthquake simulations involve large domain nonlinear time-evolution problems with locally complex structures. For example, soft soil near surface have complex geometry and softens during strong ground motion under large earthquakes. Thus, we solve the target dynamic nonlinear continuum mechanics problem using a nonlinear dynamic 3D finite-element method with low-order solid elements because this method is suitable for modeling complex geometry and analytically satisfies the traction-free boundary condition at the surface. The target equation using Newmark-β method ($\beta = 1/4$, $\delta = 1/2$) for time integration is:

$$\mathbf{A}^n \ \delta\mathbf{u}^n = \mathbf{b}^n, \tag{1}$$

where

$$\begin{cases} \mathbf{A}^n = \frac{4}{dt^2}\mathbf{M} + \frac{2}{dt}\mathbf{C}^n + \mathbf{K}^n, \\ \mathbf{b}^n = \mathbf{f}^n - \mathbf{q}^{n-1} + \mathbf{C}^n\mathbf{v}^{n-1} + \mathbf{M}\left(\mathbf{a}^{n-1} + \frac{4}{dt}\mathbf{v}^{n-1}\right), \end{cases}$$

and

$$\begin{cases} \mathbf{q}^n = \mathbf{q}^{n-1} + \mathbf{K}^n\delta\mathbf{u}^n, \\ \mathbf{u}^n = \mathbf{u}^{n-1} + \delta\mathbf{u}^n, \\ \mathbf{v}^n = -\mathbf{v}^{n-1} + \frac{2}{dt}\delta\mathbf{u}^n, \\ \mathbf{a}^n = -\mathbf{a}^{n-1} - \frac{4}{dt}\mathbf{v}^{n-1} + \frac{4}{dt^2}\delta\mathbf{u}^n. \end{cases} \tag{2}$$

Here, $\delta\mathbf{u}$, $\mathbf{u}$, $\mathbf{v}$, $\mathbf{a}$, and $\mathbf{f}$ are incremental displacement, displacement, velocity, acceleration, and external force vectors, respectively. $\mathbf{M}$, $\mathbf{C}$, and $\mathbf{K}$ are the consistent mass, damping, and stiffness matrices, respectively. dt is the time increment and n is the time step. We use Rayleigh damping for $\mathbf{C}$, where the element damping matrix $\mathbf{C}_e^n$ is calculated using the element consistent mass matrix $\mathbf{M}_e$ and the element stiffness matrix $\mathbf{K}_e^n$ as $\mathbf{C}_e^n = \alpha\mathbf{M}_e + \beta\mathbf{K}_e^n$. Here, α and β are obtained by solving the following least squares equation:

$$\text{minimize} \left[\int_{f_{\min}}^{f_{\max}} \left(h - \frac{1}{2}\left(\frac{\alpha}{2\pi f} + 2\pi f\beta \right) \right)^2 \mathrm{d}f \right],$$

where, $f_{\max}$, $f_{\min}$, and h are the maximum and minimum target frequency and the damping ratio, respectively. Although arbitrary constitutive models can be used, we use the Ramberg Osgood model [9] and Masing rule [10] for the nonlinear constitutive modeling in urban earthquake simulations. In summary, time history response $\mathbf{u}^n$ is computed by repeating the following steps.

1. Read boundary conditions.
2. Evaluate $\mathbf{C}^n$ and $\mathbf{K}^n$ based on the constitutive relations and strain at time-step $n-1$.
3. Obtain $\delta\mathbf{u}^n$ by solving Eq. 1.
4. Update Eq. 2 using $\delta\mathbf{u}^n$.

Since most of the computation cost is incurred in solving Eq. 1, we explain the details of the solver in the next subsection.

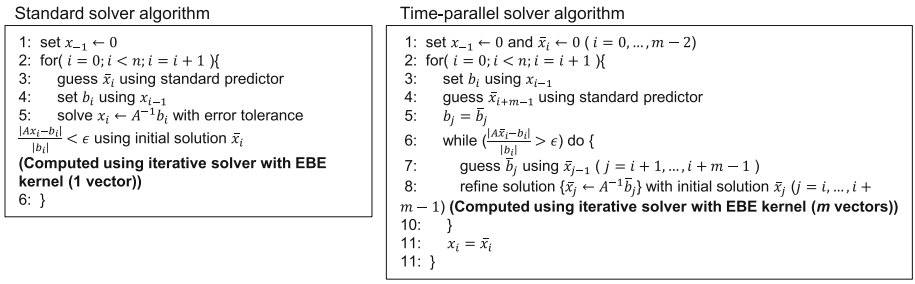

Fig. 1. Standard and time-parallel solver algorithm. The same solution is obtained within error tolerance ϵ using both of the solver algorithms.

2.2 Overview of Time-Parallel Finite-Element Solver Algorithm

Solving Eq. 1 is a common target problem arising from low-ordered implicit finite-element methods used in solid continuum mechanics problems in engineering and science fields. In these low-ordered (e.g., second-ordered) implicit simulations, iterative solvers such as Conjugate Gradient (CG) methods are generally used. Since these solvers involve much random access and intensive memory access, it is difficult to attain performance on current systems.

The time-parallel algorithm in *GHYDRA* aims to accelerate this solver by reducing random access by using the uniformity of mesh in the time domain [8]. As shown in Fig. 1, instead of solving one time step at a time, several time steps are solved together in parallel. When solving m time steps together, the arithmetic count per iteration of the iterative solver becomes m times larger. However, as the solutions for future time steps can be used as accurate initial solutions, the total number of iterations becomes approximately $1/m$ (see [8] for detailed comparison of convergence of this method). Thus, the total arithmetic count does not change by using this algorithm. However, since the mesh connectivity does not change in time, the random access involved in the kernel used to compute matrix-vector products can be reduced, which leads to shorter time-to-solution of the application with the same solution within the solver error threshold ϵ. This time-parallel algorithm is different from parallel-in-time integration methods intended to improve the time-to-solution at the strong scaling limits, as the *GHYDRA* algorithm uses the uniformity of mesh in the time domain to reduce

random data access and thus enables speedup of the application even within the strong scaling limits.

In *GHYDRA*, the time-parallel algorithm is combined with inexact preconditioned conjugate gradient method, multi-grid method, and mixed precision computation. Below we briefly summarize these components:

- Inexact preconditioned conjugate gradient method [11]: An inexact preconditioned conjugate gradient method is a preconditioned conjugate gradient method which uses another solver for solving the preconditioning matrix equation instead of multiplying a fixed preconditioning matrix. As it is common to use another iterative solver for solving the preconditioning matrix equation, we call the original CG loop as the "outer loop" and the preconditioning solver as the "inner loop". As the inner loop does not need to be solved exactly, this makes room for making improvements such as multi-grid and mixed precision arithmetic.
- Multi-grid method: We reduce the computation cost and communication size of the inexact preconditioned conjugate gradient method by using a multi-grid solver for the inner loop. Here we use a two step geometric multi-grid; we use the targeted second-ordered tetrahedral mesh FEmodel for the inner fine loop, and use the same mesh without intermediate nodes of each element for the inner coarse loop (i.e., a first-ordered tetrahedral mesh FEmodel_c). We use a 3×3 block Jacobi preconditioned conjugate gradient solver for solving the inner fine and inner coarse loops.
- Mixed precision arithmetic: Even for a target problem in FP64, we need only to solve the preconditioning matrix equation roughly. Thus, we compute the whole multi-grid preconditioner in FP32. This leads to halving memory access and communication size, and enable usage of high-performance FP32 arithmetic units equipped in recent CPUs.

When using the algorithm above, we can move most of the compute cost of the outer loop to the inner loops by using appropriate threshold values in the inner CG solvers. Thus, even though it may seem a complicated algorithm, the performance of the whole solver is dependent on kernels such as matrix-vector product kernels, inner product kernels, or saxpy kernels which are in common with standard CG solvers. In the following, we explain the most time consuming matrix-vector product kernel used in the inner fine loop, which is not straightforward to attain performance on many-core wide SIMD CPUs.

2.3 Baseline Element-by-Element Kernel Algorithm

As the relative memory transfer capability to floating point computation capability is becoming lower, using an algorithm that can reduce memory access is becoming important for fast time-to-solution. Thus, the Element-by-Element (EBE) method, which is a matrix-free matrix-vector multiplication method, is used for computation of $\mathbf{f} = \mathbf{Au}$. Here, $\mathbf{u}, \mathbf{f}$ are displacement and nodal force vectors, and $\mathbf{A}$ is the global matrix that is generated by superimposing element

matrices $\mathbf{A}_i^e$. $\mathbf{A}_i^e$ can be computed by using coordinates of nodes ($\mathbf{x}$) and element material properties. In standard matrix-vector product methods, $\mathbf{f} = \mathbf{A}\mathbf{u}$ is computed by reading the global matrix $\mathbf{A}$ from memory and multiplying it with $\mathbf{u}$. In the contrary, in the EBE method, matrix-vector products are computed by computing local matrix-vector products

$$\mathbf{f}_i^e = \mathbf{A}_i^e \mathbf{u}_i^e = \mathbf{A}_i^e \mathbf{Q}_i^{e\mathrm{T}} \mathbf{u}, \tag{3}$$

and adding them up as

$$\mathbf{f} = \sum_i \mathbf{Q}_i^e \mathbf{f}_i^e. \tag{4}$$

Here, $\mathbf{Q}_i^e$ are matrices for mapping element-wise nodal values to global nodal values. Since the coordinates, displacement and force vectors $\mathbf{x}, \mathbf{u}, \mathbf{f}$ can be kept on cache by reordering of nodes and elements, we can drastically reduce memory access when compared with methods reading the global matrix from memory. On the other hand, the EBE method involves more computation and consequently transfers memory access cost to computation cost.

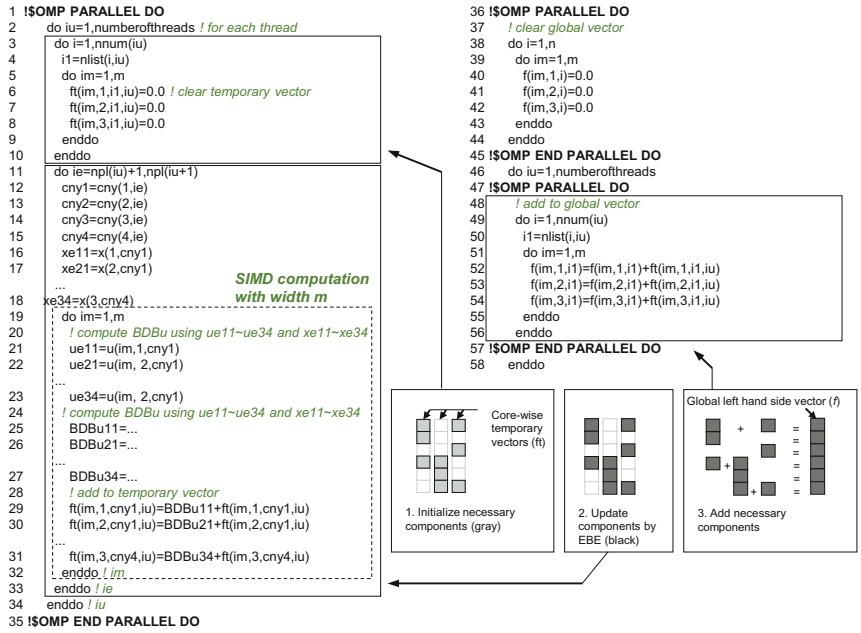

Fig. 2. Baseline EBE kernel with m vectors

Although the EBE method has low algorithmic Byte/FLOP and is potentially suitable for current computers, it is not straightforward for parallel computation due to the data recurrence for adding local force vectors of elements with shared nodes in Eq. 4. Figure 2 shows the multi-core EBE computation

algorithm for *GHYDRA*. Here, temporary vectors `ft` are allocated for each core, initialized on lines 3–10, and the core wise results are added to `ft` in lines 11–33. Finally, the core-wise results are added to the global vector `f` in lines 46–58. The innermost loop for time-parallelism (m) can be computed using SIMD units. However, in practice, we use $m = 4$ in earthquake problems as increasing m leads to deterioration of the accuracy of the predicted solutions and thus increase in the total number of arithmetic counts. Thus, only 1/4 of the SIMD lanes of 512 bit SIMD registers (i.e., 4 out of the 16 FP32 lanes) can be used. Although this is better than the non-time-parallel algorithm in which SIMD cannot be used completely due to the data recurrence involved in lines 29–31, this EBE implementation is expected to lead to poor performance on recent many-core CPUs with wider SIMD units. Development of EBE kernel algorithms that uses SIMD in full width is required for improving performance.

```
1  !$OMP PARALLEL DO
2    do iu=1,numberofthreads ! for each thread
3      do i=1,nnum(iu)
4        i1=nlist(i,iu)
5        do im=1,m
6          ft(im,1,i1,iu)=0.0 ! clear temporary vector
7          ft(im,2,i1,iu)=0.0
8          ft(im,3,i1,iu)=0.0
9        enddo
10   enddo
11   ! block loop with blocksize NL/m
12   do ieo=npl(iu)+1,npl(iu+1),NL/m
13     ! load ue, xe
14     do ie=1,min(NL/m,npl(,iu+1)-ieo+1)
15       cny1=cny(1,ieo+ie-1)
16       cny2=cny(2,ieo+ie-1)
17       cny3=cny(3,ieo+ie-1)
18       cny4=cny(4,ieo+ie-1)
19       do im=1,m
20         ue11(im+(ie-1)*m)=u(im,1,cny1)      SIMD
21         ue21(im+(ie-1)*m)=u(im,2,cny1)      (width=m)
         ...                                   computation
22         ue34(im+(ie-1)*m)=u(im,3,cny4)
23         xe11(im+(ie-1)*m)=x(1,cny1)
24         xe21(im+(ie-1)*m)=x(2,cny1)
         ...
25         xe34(im+(ie-1)*m)=x(3,cny4)
26       enddo
27     enddo
```

```
28     ! compute BDBu
29     do i=1,NL
30       BDBu11(ie)=...
31       BDBu21(ie)=...          SIMD computation
       ...
32       BDBu34(ie)=...
33     enddo
34     ! add to global vector
35     do ie=1,min(NL/m, npl(icolor,iu+1)-ieo+1)
36       cny1=cny(1,ieo+ie-1)
37       cny2=cny(2,ieo+ie-1)                  SIMD
38       cny3=cny(3,ieo+ie-1)                  (width=m)
39       cny4=cny(4,ieo+ie-1)                  computation
40       do im=1,m
41         ft(im,1,cny1,iu)=BDBu11(im+(ie-1)*m)+f(im,1,cny1,iu)
42         ft(im,2,cny1,iu)=BDBu21(im+(ie-1)*m)+f(im,2,cny1,iu)
       ...
43         ft(im,3,cny4,iu)=BDBu34(im+(ie-1)*m)+f(im,3,cny4,iu)
44       enddo
45     enddo
46   enddo ! ieo
47 enddo ! iu
48 !$OMP END PARALLEL DO
49   Add ft in to f (same as lines 36-58 of Fig. 2)
```

Fig. 3. EBE kernel with m vectors for wide-SIMD CPUs

3 Developed EBE Kernel Algorithms for Many-Core Wide SIMD CPUs

In this section, we show the developed algorithms for faster EBE computation on many-core wide SIMD CPUs.

We first develop an algorithm to utilize SIMD units in the main computation loop (Fig. 3). As the data recurrence in addition of results into the `ft` vector was blocking the use of SIMD units, we split the innermost loop into two loops; i.e., the computation part of $\mathbf{f}_i^e = \mathbf{A}_i^e \mathbf{u}_i^e$ (Fig. 3 lines 14–33) and the summation part of $\mathbf{f}_i^e$ to $\mathbf{f}$ (lines 35–45). This leads to use of SIMD operations to the computationally rich first loop. Here, we reduce the overhead of loop splitting by

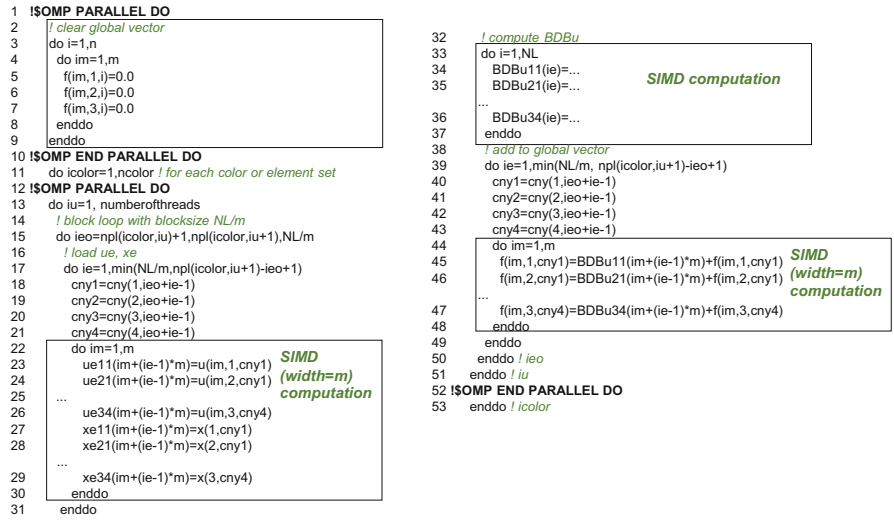

Fig. 4. Coloring/thread partitioning of EBE kernel with m vectors for wide-SIMD CPUs. The same code can be used for both coloring/thread partitioning methods shown in Fig. 5.

blocking the loop with small block size (`NL`, which is typically set to the SIMD vector length). This keeps the temporary buffers `BDBu11-34` on cache.

When using many cores, the size of the thread-wise temporary buffer `ft` becomes large. In addition, the overhead of initializing `ft` and adding the thread-wise results to the global vector `f` is not negligible in the total kernel cost. Thus, we developed a thread partitioning method to eliminate the use of thread-wise temporary buffers (Fig. 4). The procedure in standard coloring methods are to color the whole mesh such that elements in each color does not have shared nodes (Fig. 5a). Instead, we partition the domain with prescribed thread numbers (Fig. 5b). Here, we partition the overall mesh into the number of threads using a graph partitioning method (METIS [12]). Then, we remove elements that share nodes between the thread partitions. The remaining elements becomes the first set of elements to be computed (Set #1 of Fig. 5b). The removed elements are partitioned again to recursively decompose the mesh into sets (Set #2 and #3 of Fig. 5b). As will be shown in Sect. 4, the proposed thread partitioning method enables better cache reuse of nodal values `u`, `x` and `f` when compared with the case of standard coloring methods.

In practice, we use $m = 4$ in earthquake problems as increasing m leads to deterioration of the accuracy of the predicted solutions and thus increase in the total number of arithmetic counts. On the other hand, 16 FP32 floats can be packed in a single 512 bit SIMD register; thus, we can only use the first 1/4 of the SIMD capability in the vector load/store computation. In order to accelerate this part, we used 16 wide SIMD for loading from `u` while using 4 wide SIMD for storing to local vectors `ueXX` (Fig. 6a). Similarly, we used 4 wide SIMD for

loading local vectors `BDBuXX` while using 16 wide SIMD for loading and storing `f` (Fig. 6b). This enables reduction of inefficient 4 wide SIMD accesses and thus improvement in overall kernel performance.

4 Performance Measurements

We measure performance of the developed kernels and the accelerated finite-element application on the K computer, Intel Xeon Phi Knights Landing based Oakforest-PACS, and an Intel Skylake Xeon Gold CPU based system. The latter two systems are examples of many-core systems with 512-bit SIMD. Table 1 summarizes the configuration of the systems. K computer (K) [7] consists of 82,944 compute nodes, each with a single eight-core SPARC64 VIIIfx CPU. Each core has two sets of SIMD FMA arithmetic units of width 2. Oakforest-PACS (OFP) [13] is a supercomputer system introduced by the Joint Center for Advanced HPC, which was established by the University of Tokyo and the University of Tsukuba. The system comprises 8,208 nodes with a 68-core Intel Xeon Phi 7250 (Knights Landing) CPU [14], 96 GB of DDR4 RAM, and 16 GB of stacked 3D MCDRAM. Intel Skylake Xeon Gold CPU based system comprises two 20-core Intel Skylake Xeon Gold 6148 CPU [15] and 192 GB of DDR4 RAM.

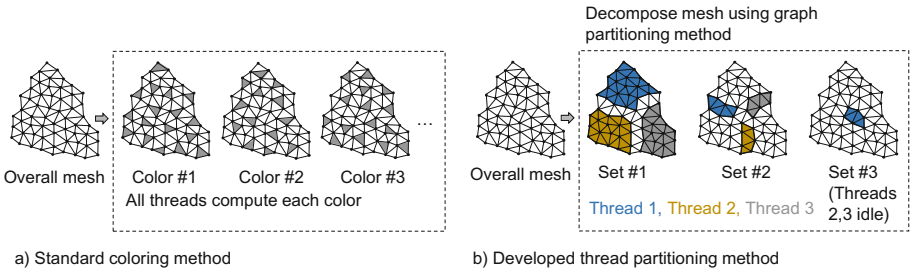

Fig. 5. Coloring/thread partitioning methods for the EBE kernel

Table 1. Performance measurement environment. Memory bandwidth on OFP is measured values of STREAM benchmark, while others are hardware peak values.

	K computer	Oakforest-PACS	Intel Skylake Xeon Gold based server
Nodes	8	1	1
Sockets/node	1	1	2
Cores/socket	8	68	20
FP32 SIMD width	2	16	16
Clock frequency	2.0 GHz	1.4 GHz	2.4 GHz
Total peak FP32 FLOPS	1024 GFLOPS	6092 GFLOPS	6144 GFLOPS
Total DDR bandwidth	512 GB/s	80.1 GB/s	255.9 GB/s
Total MCDRAM bandwidth	-	490 GB/s	-

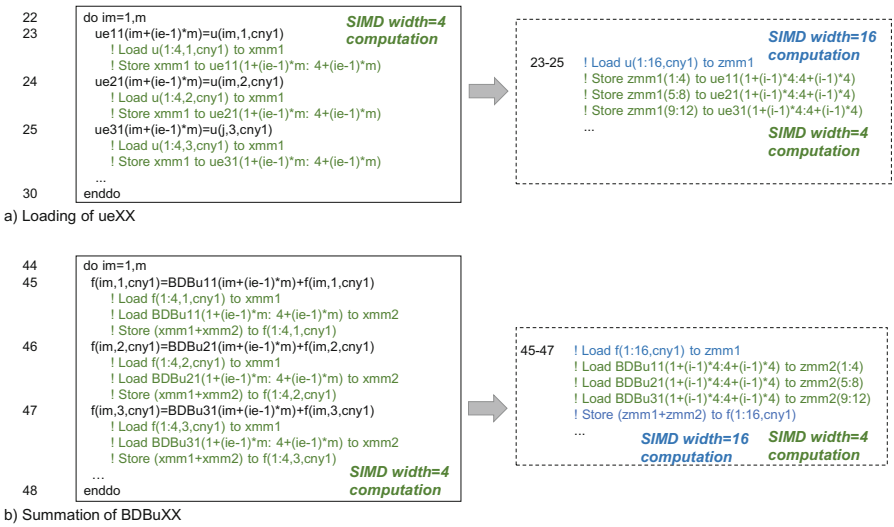

Fig. 6. Tuning of m vector EBE kernel for 512 bit SIMD architecture. Here, `xmm` indicate 128 bit FP32 registers and `zmm` indicate 512 bit FP32 registers.

Both the Xeon Phi 7250 CPU and Skylake Xeon Gold 6148 CPU support AVX-512 SIMD instructions.

We first measure performance of the EBE kernel. For the K computer, we use 8 nodes each with 1 MPI process with 8 OpenMP threads per process. For OFP, we use quadrant/flat mode, and run 8 MPI processes with 8 OpenMP threads per process on a single node. Each thread is bound to one CPU core (hyperthreading is not used), and we use `numactl --preferred=1` option such that memory is preferentially allocated to MCDRAM. For the Skylake system, we use 8 MPI processes with 5 OpenMP threads per process on a single node. Each MPI process on K/OFP/Skylake runs on a mesh with 6,534,144 second-order tetrahedral elements and 9,044,560 nodes and elapsed time for computing 529 times of matrix-vector products is measured. Figure 7 shows the EBE kernel performance. Comparing the K computer and OFP for the baseline kernel with one vector, we can see that high performance of 26.1% FP32 peak efficiency is attained for the K computer; however, only 1.98% is attained on OFP. The kernel performance is improved by using the time-parallel algorithm with $m = 4$; a 9.45 s/7.52 s = 1.25-fold speedup was obtained on K computer and 20.1 s/7.50 s = 2.68-fold speedup was obtained on OFP, respectively, for elapsed time per vector. This performance is expected to be further improved by using the developed EBE algorithms. First we see the effectiveness of the kernel algorithm enabling use of SIMD for the main computation part. By using SIMD by loop splitting and blocking for the main computation part of the EBE kernel, we can shorten elapsed time from 7.50 s to 3.75 s on OFP. Note that loop splitting and blocking is disabled for the K computer as the 2 wide FP32 SIMD of

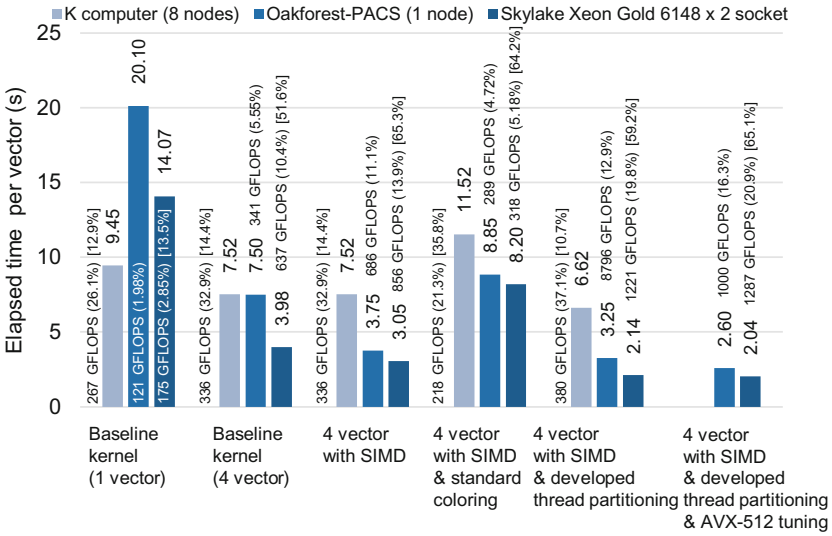

Fig. 7. EBE kernel performance. Elapsed time per vector is shown. Numbers in brackets indicate efficiency to FP32 peak, and numbers in square brackets indicate memory bandwidth efficiency to hardware peak of K computer and Skylake system. 8 nodes of the K computer (1 MPI process × 8 OpenMP threads per node), 1 node of OFP (8 MPI processes × 8 OpenMP threads), and a Skylake system (8 MPI processes × 5 OpenMP threads) is used for computation.

the K computer can be directly applied to the innermost time-parallel m loop. Next we see the effectiveness of using coloring and thread aware partitioning. When using standard coloring generated by serial greedy algorithm, the mesh was decomposed into 41 colors, resulting in total of 263.9 GB memory transfer per node incurred for computation of the EBE kernel on the K computer. This is more memory transfer than the baseline algorithm with 69.3 GB memory transfer, as the cache reuse for `u`, `x` and `f/ft` is disabled; which lead to longer elapsed time. When using the developed thread partitioning method, the mesh was decomposed into 5 colors. With more cache reuse, the total memory transfer was reduced to 45.3 GB, which resulted to 7.52 s/6.62 s = 1.13-fold speedup on the K computer and 3.75 s/3.25 s = 1.15-fold speedup on OFP. Furthermore, by using the AVX-512 tuning for reducing random data access, the elapsed time was decreased to 2.60 s on OFP. This lead to high performance of 16.3% of peak FP32 performance on 64 cores of OFP's Intel Xeon Phi Knights Landing CPU. We can see that the effective use of SIMD and circumventing random access to temporary vectors lead to significant speedup of 7.50 s/2.60 s = 2.88-fold from the baseline algorithm with $m = 4$ vectors on OFP. Compared with the baseline algorithm without time-parallelism ($m = 1$), this is 20.1 s/2.60 s = 7.73-fold speedup on OFP. The developed EBE algorithm is also effective for the Skylake system, and lead to 3.98 s/2.04 s = 1.95-fold from the baseline algorithm with

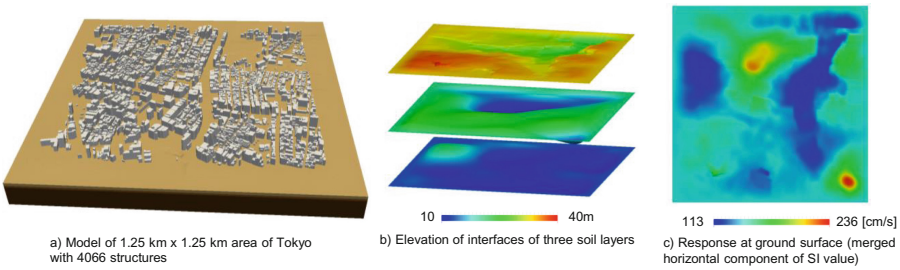

Fig. 8. Urban earthquake problem settings and computation results. Earthquake wave propagation in the three-layered soil structure is computed in this application example.

$m = 4$ vectors. Compared with the baseline algorithm without time-parallelism ($m = 1$), this is 14.1 s/2.04 s = 6.89-fold speedup on the Skylake system.

Next we see the performance of developed application on an urban earthquake problem targeting a 1.25 km × 1.25 km area of Tokyo. Using digital elevation map of [16] and soil information of [17], we constructed a soil model consisting of three soil layers. We discretized this problem with minimum element size of 1 m, which lead to a problem size of 1,022,620,536 degrees-of-freedom, 252,738,195 second-order tetrahedral elements and 340,873,512 nodes (Fig. 8a, b). We input the wave observed during 1995 Kobe Earthquake [18] with time stepping $dt = 0.01$ s. The mesh is partitioned with METIS into 1,152 partitions and computed using 144 nodes of OFP (8 MPI processes per node). Figure 9 shows the elapsed time for solving the first 25 time steps using *GHYDRA* with the baseline EBE kernel and *GHYDRA* with the developed EBE kernel (both with $m = 4$). We also show performance of *GAMERA*, which can be considered as the non-time parallel version of *GHYDRA* with the baseline EBE kernel ($m = 1$). We can see that the application was accelerated by 125.6 s/61.9 s = 2.03-fold by using the developed EBE kernel algorithms leading to high peak performance of 11.6% of FP64 peak. Together with the 247.2 s/125.6 s = 1.97-fold speedup using time-parallelism, the application was accelerated by total of 3.99-fold when compared

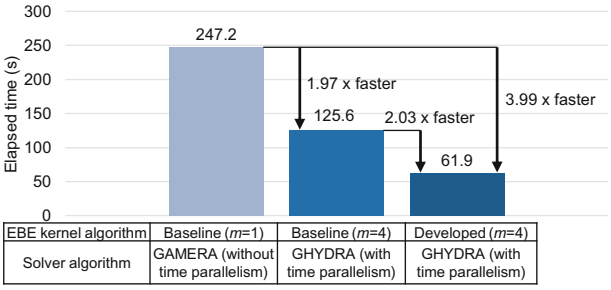

Fig. 9. Elapsed time for solving the first 25 time steps of application problem. Measured on 144 nodes of OFP.

to the SC14 Gordon Bell Prize Finalist solver *GAMERA*. We can see that the acceleration of the EBE kernel with algorithms suitable for wide SIMD and multi-cores are effective for attaining high performance and shorter time-to-solution of the overall application. With less energy-to-solution, we can afford to conduct more detailed simulations of larger areas of interest, which is expected to contribute to earthquake disaster mitigation.

5 Closing Remarks

In this paper we developed algorithms to accelerate the Element-by-Element kernel in unstructured low-order implicit finite-element methods on systems with many-core wide SIMD CPUs. By using the developed algorithm on the Intel Xeon Phi Knights Landing based Oakforest-PACS system, the elapsed time of the EBE kernel and total unstructured finite-element application was accelerated by 2.88-fold and 2.03-fold, respectively. The developed EBE kernel algorithms were also effective for the K computer and an Intel Skylake Xeon Gold CPU based system. These insights are expected to enable high performance on other large-scale many-core wide CPU based supercomputer systems, enabling energy efficient finite-element computation towards exascale computing.

Acknowledgments. Our results were obtained using the Oakforest-PACS at the Joint Center for Advanced HPC and the K computer at the Center for Computational Science, RIKEN (hp170249, hp180217). We acknowledge support from the Japan Society for the Promotion of Science (17K14719, 26249066, 25220908, 18H05239).

References

1. Outline of the Development of the Post-K computer. https://www.r-ccs.riken.jp/en/postk/project/outline
2. Rudi, J., et al.: An extreme-scale implicit solver for complex PDEs: highly heterogeneous flow in earth's mantle. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC 2015), p. 5. ACM (2015)
3. Yang, C., et al.: 10M-core scalable fully-implicit solver for nonhydrostatic atmospheric dynamics. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC 2016), p. 6. IEEE Press (2016)
4. Ichimura, T., et al.: Implicit nonlinear wave simulation with 1.08T DOF and 0.270T unstructured finite elements to enhance comprehensive earthquake simulation. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC 2015), p. 4. ACM (2015)
5. Ichimura, T., et al.: Physics-based urban earthquake simulation enhanced by 10.7 BlnDOF x 30 K time-step unstructured FE non-linear seismic wave simulation. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC 2014), pp. 15–26. IEEE Press (2014)

6. Winget, J.M., Hughes, T.J.: Solution algorithms for nonlinear transient heat conduction analysis employing element-by-element iterative strategies. Comput. Methods Appl. Mech. Eng. **52**(1–3), 711–815 (1985)
7. Miyazaki, H., Kusano, Y., Shinjou, N., Shoji, F., Yokokawa, M., Watanabe, T.: Overview of the K computer system. FUJITSU Sci. Tech. J. **48**(3), 302–309 (2012)
8. Ichimura, T., et al.: A fast scalable implicit solver with concentrated computation for nonlinear time-evolution problems on low-order unstructured finite elements. In: IEEE International Parallel and Distributed Processing Symposium (IPDPS), Vancouver, BC 2018, pp. 620–629 (2018). https://doi.org/10.1109/IPDPS.2018.00071
9. Idriss, I.M., Dobry, R., Sing, R.D.: Nonlinear behavior of soft clays during cyclic loading. J. Geotech. Eng. Div. **104**(ASCE 14265) (1978)
10. Masing, G.: Eigenspannungen und Verfestigung beim Messing. In: Proceedings of the 2nd International Congress of Applied Mechanics, pp. 332–335 (1926)
11. Golub, G.H., Ye, Q.: Inexact conjugate gradient method with inner-outer iteration. SIAM J. Sci. Comput. **21**(4), 1305–1320 (1997)
12. Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J. Sci. Comput. **20**(1), 359–392 (1998)
13. Oakforest-PACS. http://www.cc.u-tokyo.ac.jp/system/ofp/index-e.html
14. Sodani, A.: Knights landing (KNL): 2nd Generation IntelR Xeon Phi processor. In: IEEE Hot Chips 27 Symposium (HCS), Cupertino, CA, pp. 1–24 (2015). https://doi.org/10.1109/HOTCHIPS.2015.7477467
15. Kumar, A.: The New Intel Xeon Processor Scalable Family (Formerly Skylake-SP). In: 2017 IEEE Hot Chips 29 Symposium (HCS), Cupertino, CA (2017)
16. 5m mesh digital elevation map, Tokyo ward area, Geospatial Information Authority of Japan. https://www.gsi.go.jp/MAP/CD-ROM/dem5m/index.htm
17. National Digital Soil Map, The Japanese Geotechnical Society. https://www.jiban.or.jp/?page_id=432
18. Strong ground motion of The Southern Hyogo prefecture earthquake in 1995 observed at Kobe JMA observatory, Japan Meteorological Agency. https://www.data.jma.go.jp/svd/eqev/data/kyoshin/jishin/hyogo_nanbu/dat/H1171931.csv