Design of Parallel BEM Analyses Framework for SIMD Processors

Tetsuya Hoshino[1], Akihiro Ida[1], Toshihiro Hanawa[1], and Kengo Nakajima[1]

Information Technology Center, The University of Tokyo, Tokyo, Japan
hoshino@cc.u-tokyo.ac.jp, ida@cc.u-tokyo.ac.jp, hanawa@cc.u-tokyo.ac.jp, nakajima@cc.u-tokyo.ac.jp

Abstract. Parallel Boundary Element Method (BEM) analyses are typically conducted using a purpose-built software framework called BEM-BB. This framework requires a user-defined function program that calculates the i-th row and the j-th column of the coefficient matrix arising from the convolution integral term in the fundamental BEM equation. Owing to this feature, the framework can encapsulate MPI and OpenMP hybrid parallelization with $\mathcal{H}$-matrix approximation. Therefore, users can focus on implementing a fundamental solution or a Green's function, which is the most important element in BEM and depends on the targeted physical phenomenon, as a user-defined function. However, the framework does not consider single instruction multiple data (SIMD) vectorization, which is important for high-performance computing and is supported by the majority of existing processors. Performing SIMD vectorization of a user-defined function is difficult because SIMD exploits instruction-level parallelization and is closely associated with the user-defined function. In this paper, a conceptual framework for enhancing SIMD vectorization is proposed. The proposed framework is evaluated using two BEM problems, namely, static electric field analysis with a perfect conductor and static electric field analysis with a dielectric, on Intel Broadwell (BDW) processor and Intel Xeon Phi Knights Landing (KNL) processor. It offers good vectorization performance with limited SIMD knowledge, as can be verified from the numerical results obtained herein. Specifically, in perfect conductor analyses conducted using the $\mathcal{H}$-matrix, the framework achieved performance improvements of 2.22x and 4.34x compared to the original BEM-BB framework for the BDW processor and KNL, respectively.

1 Introduction

The boundary element method (BEM) has several scientific applications. This method requires fewer unknowns and has a lower meshing cost compared to other volume discretization methods because it requires only the surface of the target objects for analysis. However, the computational cost and memory footprint of BEM analysis are significantly high because a dense coefficient matrix is generated during the analysis. To overcome these problems, parallel computing and approximation techniques, such as hierarchical matrices ($\mathcal{H}$-matrices) [1–3],

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_46

$\mathcal{H}^2$-matrices [4], and the fast multipole method (FMM) [5] are often used for BEM analysis. Although these techniques have huge programming costs, BEM-BB [6], an open-source software framework for parallel BEM analysis, is useful to for reducing these costs. The framework employs $\mathcal{H}$-matrices to approximate the dense coefficient matrix, and it is parallelized using the MPI and OpenMP models. The BEM-BB framework allows for faster BEM analysis on parallel computers by simply preparing programs to calculate the integrals of boundary elements, settings of boundary conditions, and analysis output. In addition, the parallelization and the approximation programs are encapsulated in the framework. Thus, users can concentrate on developing the most important aspects of BEM analysis, namely, a user-defined function for calculating the i-th row and the j-th column of the coefficient matrix. Furthermore, the user-defined function may vary depending on the targeted physical phenomena.

However, this framework does not consider single instruction multiple data (SIMD) vectorization, which is important for achieving high-performance computing on existing processors. For example, the most recent Intel processors, such as Skylake EP/EX and Xeon Phi Knights Landing (KNL), support AVX-512, that is, a 512-bit SIMD instruction set. SIMD vectorization cannot be separated from user-defined functions, unlike in MPI and OpenMP parallelization, because SIMD vectorization is instruction-level parallelization and because user-defined functions can vary. However, SIMD vectorization is difficult for application programmers because it requires knowledge of the compiler and the target processor architecture.

In this paper, we present a framework design based on BEM-BB for SIMD vectorization. A design to encapsulate SIMD-related aspects is proposed. In addition, we evaluate the performance of the proposed framework by solving two problems, namely, static electric field analysis with a perfect conductor and static electric field analysis with a dielectric, which contain different user-defined functions, on Intel Broadwell processor (BDW) and Intel Xeon Phi Knights Landing (KNL). We compare the performance of the proposed framework with the original framework and that of hand-tuned user functions. The results show that the proposed framework offers performance improvements of 2.22x and 4.34x compared to the original framework for the BDW processor and the KNL processor, respectively. Furthermore, the experimental results demonstrate that the performance of the framework is comparable to that achieved using the hand-tuned programs

The remainder of this paper is organized as follows. In Section 2, we provide an overview of the BEM-BB framework. The proposed framework is described in Section 3. Numerical experiments involving electric field analysis are described in Section 4, and a few conclusions and suggestions for future work are presented in Section 6.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_46

2 BEM-BB framework

In this section, the BEM-BB framework, which is the baseline implementation in this study, is introduced. The BEM-BB software framework is used for parallel BEM analysis. It is implemented in the Fortran90 programming environment and parallelized using the OpenMP + MPI hybrid programming model. To reduce the computational cost of parallel programming, the framework supports model data input, assembly of the coefficient matrix, and solution of linear systems, steps that are generally required in BEM analysis. When employing this framework, users are required to generate user-defined functions that calculate each element of the coefficient matrix. In other words, users are required to implement a program to calculate the integrals of boundary elements, which depend on the governing target of BEM analysis. The target integral equation of the BEM-BB framework is described as follows. For $f \in H'$, $u \in H$ and a kernel function of a convolution operator $g : \mathbb{R}^d \times \Omega \to \mathbb{R}$,

$$\int_{\Omega} g(x,y)u(y)\mathrm{d}y = f \tag{1}$$

where $\Omega \subset \mathbb{R}^d$ denotes a $(d-1)$-dimensional domain, H the Hilbert space of functions on a Ω, and H' dual space of H. To numerically calculate Eq.(1), we divide the domain, Ω, into the elements $\Omega_h = \{\omega_j : j \in J\}$, where J is an index set. In weighted residual methods, such as the Ritz-Galerkin method and the collocation method, the function u is approximated from a n-dimensional subspace $H^h \subset H$. Given a basis $(\varphi_i)_{i\in\beth}$ of H^h for an index set $\beth := \{1,\ldots,N\}$, the approximant $u^h \in H^h$-u can be expressed using a coefficient vector $\phi = (\phi_i)_{i\in\beth}$ that satisfies $u^h = \sum_{i\in\beth}\phi_i\varphi_i$. Note that the supports of the basis $\Omega^h_{\varphi_i} :=$ supp φ are assembled from the sets ω_j. Equation (1) is then reduced to the following system of linear equations.

$$A\phi = b \tag{2}$$

$$A_{ij} = \int_{\Omega}\varphi_i(x)\int_{\Omega} g(x,y)\varphi(y)\mathrm{d}y\mathrm{d}x \tag{3}$$

$$b_i = \int_{\Omega}\varphi_i(x)f\mathrm{d}x \tag{4}$$

Here, $i, j \in \beth$. The user-defined function required to calculate the elements of the i-th row and the j-th column of the coefficient matrix is expressed as Eq.(3).

There are two versions of the implementation: one based on dense matrix computations and the other based on $\mathcal{H}$-matrix computations. Although the $\mathcal{H}$-matrix version depends on the distributed parallel $\mathcal{H}$-matrix library $\mathcal{H}$ACApK [7], the problems of vectorization are similar. As shown in Fig. 1, the proposed framework consists of three components: model data input, coefficient matrix generation, and linear solver. In this study, the objective is to interface coefficient matrix generation with user-defined function. Therefore, we focus on the coefficient matrix generation component.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_46

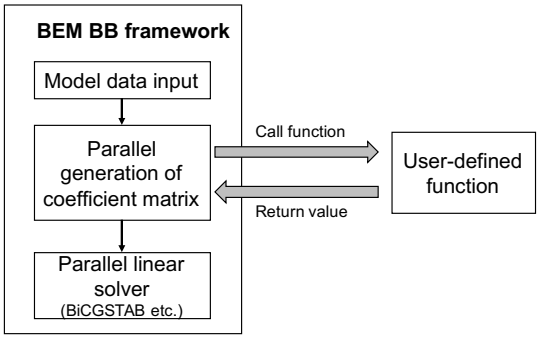

Fig. 1. The design of BEM-BB framework

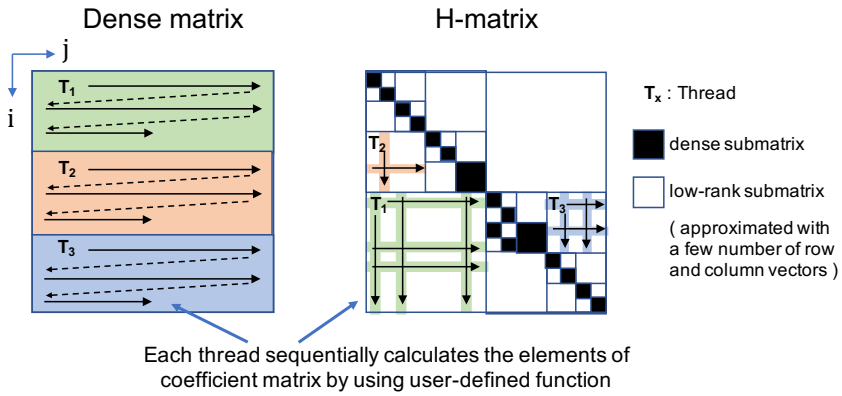

Fig. 2. Parallel generation of coefficient dense matrix and $\mathcal{H}$-matrix.

Fig. 2 shows the coefficient matrix generation part. The target coefficient matrix is distributed to multiple thread and each thread sequentially calculates the i-th row and the j-th column element by using user-defined function. The coefficient matrices generated using the dense matrix version and the $\mathcal{H}$-matrix version are a dense matrix and an $\mathcal{H}$-matrix, respectively. A $\mathcal{H}$-matrix is also called a hierarchical matrix. $\mathcal{H}$-matrices are among the techniques used to approximate dense matrices. An $\mathcal{H}$-matrix is a set of low-rank approximated sub-matrices and small dense sub-matrices as shown in Fig. 2. $\mathcal{H}$ACApK generates the coefficient $\mathcal{H}$-matrix by exploiting the user-defined function according to the Adaptive Cross Approximation (ACA) algorithm [9]. The ACA algorithm is an approximation technique used to generate a low-rank approximated matrix of a dense matrix without generating the target dense matrix.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_46

```
real(8) function ppohBEM_matrix_element_ij(i,j,nond,nofc,nond_on_fc,np,
    intpara_fc,nint_para_fc,dble_para_fc,ndble_para_fc,face2node)
  !$omp declare simd
  type :: coordinate
    real(8) :: x,y,z
  end type coordinate
  integer ,intent(in) :: i,j,nond,nofc,nond_on_fc,nint_para_fc,
      ndble_para_fc
  type(coordinate),intent(in) :: np(*)
  integer, intent(in) :: face2node(3,*),int_para_fc(nint_para_fc,*)
  real(8), intent(in) :: dble_para_fc(ndble_para_fc,*)

  ! User defined calculations for the i-th row and the j-th column
      element

end function ppohBEM_matrix_element_ij
```

Fig. 3. An interface of a user-defined function to calculate the i-th row and the j-th column element of the coefficient matrix. The function arguments after i and j are used as input variable of the calculation.

```
do i=lhp, ltp
  !$omp simd
  do j=j_st, j_en
    a(j,i) = ppohBEM_matrix_element_ij( i, j, nond, nofc, &
                                        nond_on_fc, np, intpara_fc, &
                                        nint_para_fc, dble_para_fc, &
                                        ndble_para_fc, face2node )
  enddo
enddo
```

Fig. 4. User-defined function caller for dense matrix. Here, a(j,i) is a coefficient dense matrix. The ranges of i and j are assigned to each thread adequately.

The interface of the user-defined function is shown in Fig. 3. In both versions, the function is called from each thread concurrently. To vectorize the user-defined function, the caller of the function, too, is important. Figures 4 and 5 show the callers of the user-defined functions of the dense matrix version and the $\mathcal{H}$-matrix version, respectively. Both programs call the user-defined function in loop structures. These loops are the target of SIMD vectorization. In the following sections, we treat the implementation shown in Fig. 4 as the baseline.

3 Framework Design for SIMD Vectorization with OpenMP SIMD Directives

In general, three methods are used to perform SIMD vectorization: (1) relying on compiler auto-vectorization, (2) using compiler directives, and (3) using intrinsic functions. However, vectorization using intrinsic functions is cumbersome job, and the required intrinsic functions depend completely on the user-defined

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_46

```
if( column vector calculation )
  i = ip + nstrtl-1
  !$omp simd private(j)
  do ii=1,s_m
    if(colmsk(ii)==0) then
      j = ii + nstrtt-1
      colvec(ii)=HACApK_entry_ij(i,j,st_bemv)
    endif
  enddo
else if( row vector calculation )
  j = ip + nstrtt-1
  !$omp simd private(i)
  do ii=1,t_m
    if(rowmsk(ii)==0) then
      i = ii + nstrtl-1
      rowvec(ii)=HACApK_entry_ij(i,j,st_bemv)
    endif
  enddo
endif
```

Fig. 5. User-defined function caller for sub-matrix of $\mathcal{H}$-matrix. Here, HACApK_entry_ij is a wrapper function of ppohBEM_matrix_element_ij. The structure st_bemv contains the variables required as arguments of the user-defined function.

function. In this study, we employ compiler auto-vectorization and the directive method. To use SIMD instructions efficiently, there are two constraints on the SIMD target vectors.

- There should be no data dependency among the elements of the target vector.
- Vector elements should be stored contiguously.

In addition, to generate efficient code by using compiler vectorizations, the code should be obviously vectorizable from the compiler's view point. Any new framework design should consider the above points. Furthermore, the design should be user-friendly. Efficiently vectorized SIMD code should be generated if users are unaware of compiler requirements.

3.1 New interface definition for compiler vectorization

According to the two compiler requirements, the main problem associated with vectorization pertains to data access. Even though the computations associated with a user-defined function can be executed independently, if a compiler detects possibilities of data dependency, it conservatively generates instructions that are not fully vectorized. Therefore, we propose to handle data access and computation separately in the proposed framework design. We introduce two new interfaces `set_args` (Fig. 6) and `vectorize_func` (Fig. 7) for data access and computation, respectively. Figure 8 shows the function caller based on Fig. 4. The variables `SIMDLENGTH`, which appear in Figs. 7 and 8 and are defined by users, represent the SIMD length of the target processor. For example, the recommended `SIMDLENGTH` for KNL, which has a 512-bit (= sizeof(double)

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_46

```
1  subroutine set_args(i,j,nond,nofc,nond_on_fc,np,intpara_fc,nint_para_fc,
       dble_para_fc,ndble_para_fc,face2node,darg1,darg2,...,dargN,iarg1,
       iarg2,...,iargM)
2    real(8), intent(out) :: darg1,darg2,...,dargN
3    integer, intent(out) :: iarg1,iarg2,...,iargM
4
5    !  User defined data access for calculating an element of the i-th row
         and the j-th column from arrays to scalar args
6
7  end subroutine set_args
```

Fig. 6. New interface for data access. The former arguments are the same as ppohBEM_matrix_element_ij. The latter arguments are the scalar variables used in vectorize_func. The number of arguments depends on the target application.

×8) wide SIMD unit, is 8. From the compiler's viewpoint, the `!$omp simd` loop (Fig. 8 line 14) has no data dependency because the arguments and the return values of `vector_func` have no alias and are accessed independently for each iteration of the loop. In addition, the arguments and return values are stored contiguously. At this point, if the SIMD interface of the `vectorize_func` corresponds to the SIMD length, the loop (Fig. 8 lines 13-17) is vectorized similarly to a vector function.

To safely vectorize `vectorize_func`, we constrain the function such that it cannot contain globally accessible variables, allocatable arrays, or save variables. In addition, the SIMD interfaces of all functions or subroutines called from `vectorize_func` should correspond to the SIMD length. This parallelization method is similar to the Single Program Multiple Data (SPMD) programming model because each SIMD element executes a single program simultaneously.

To reduce the data access cost, we introduce a pair of interfaces `set_args_i` and `set_args_j`. In BEM analysis, the required data such as coordinate of the i-th element and the j-th element usually depends only on the variables `i` and `j`, respectively. Therefore, the subroutines `set_args_i` and `set_args_j` are used to set arguments depending only on `i` and `j`, respectively. The pair of interfaces work effectively in the $\mathcal{H}$-matrix version. As shown in Fig. 5, `i` and `j` are constants in the lines 4-9 loop and lines 13-18 loop, respectively.

3.2 Using the framework

The new interfaces are easy to vectorize for compilers, but they are not user-friendly. Specifically, the numbers of arguments of the `set_args` subroutine and the `vectorize_func` function depend on the target application, which means users are required to modify the framework program in order to add variable declarations and correspond to the interface. In addition, users must vectorize the user-defined functions by using `!$omp declare simd` pragma. Furthermore, if users insert a wrong directive, the compiler generates a correct but unvectorized slow executable, which is often more cumbersome compared to a bug.

To minimize these difficulties, we require users to prepare the followings.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_46

```
real(8) function vectorize_func(darg1,darg2,...,dargN,iarg1,iarg2,...,
    iargM)
  !$omp declare simd simdlen(SIMDLENGTH)
  real(8), intent(in) :: darg1,darg2,...,dargN
  integer, intent(in) :: iarg1,iarg2,...,iargM

  ! User defined calculations for an element of the i-th row and j-th
      column

end function vectorize_func
```

Fig. 7. New calculation interface. This function should be called after the set_args subroutine and vectorized. All arguments of this function should have intent(in) attribute.

- Implement include files.
- Implement the `set_args, set_args_i, set_args_j` and the `vectorize_func` without the SIMD directives in the file "user_func.f90".
- Correctly implement the dummy function `ppohBEM_matrix_element_ij_dummy` (Fig. 9) without modifying the dummy function itself.
- Provide `SIMDLENGTH` of the target processor by using the -D compiler flag.

The include files that appear in the dummy function are used in the subroutine call interface. First, users of the framework must implement the include files as a fill-in-the-blank puzzle to correct the dummy function. In other words, the return value of the dummy function should be equal to `ppohBEM_matrix_element_ij`. At this point, users need not consider SIMD vectorization. Notably, users cannot modify the dummy function itself. If users do not need the `set_args` function, they must create an empty "call_set_args.inc" file. Second, the users must implement the user-defined functions in "user_func.f90." Notably, users need not consider SIMD vectorization as well. Finally, users must define the variable `SIMDLENGTH` by using a compiler option. During compiling, the compile script automatically inserts SIMD directives into the user-defined functions implemented in user_func.f90 and automatically transforms the include files to adjust the framework, as shown in Fig. 10. Based on the results of the auto-transformation, we succeeded in separating almost all aspects related to SIMD vectorization from the user-defined function. Therefore, users are required to set only the `SIMDLENGTH` of the target processor.

4 Numerical Evaluations

4.1 Test Model and Processors

In this section, we evaluated the proposed framework by performing BEM analysis of two electrostatic field problems. We assumed a perfectly conductive sphere and a dielectric sphere. The electric potentials of the perfect conductor and the dielectric are given by the following functionals $\mathcal{P}$ and $\mathcal{D}$, respectively:

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_46

```
real(8),dimension(SIMDLENGTH) :: ans
real(8),dimension(SIMDLENGTH) :: darg1,darg2,...,dargN
integer,dimension(SIMDLENGTH) :: iarg1,iarg2,...,iargM
  ...
do i=lhp, ltp
  do jj=j_st, j_en, SIMDLENGTH
    ii = 1
    do j=jj,min(jj+SIMDLENGTH-1,j_en)
      call set_args(i,j,...,darg1(ii),darg2(ii),...,dargN(ii) &
                   ,iarg1(ii),iarg2(ii),...,iargM(ii))
      ii = ii+1
    end do
    !$omp simd
    do ii = 1, SIMDLENGTH
      ans(ii) = vectorize_func(darg1(ii),darg2(ii),...,dargN(ii) &
                   ,iarg1(ii),iarg2(ii),...,iargM(ii))
    end do
    ii = 1
    do j=jj,min(jj+SIMDLENGTH-1,j_en)
      a(j,i) = ans(ii)
      ii = ii+1
    end do
  enddo
enddo
```

Fig. 8. User-defined function using new interface caller for dense matrix.

$$\mathcal{P}[u](x) := \int_{\Omega} \frac{1}{4\pi||x-y||} u(y)\mathrm{d}y, x \in \Omega \tag{5}$$

$$\mathcal{D}[u](x) := \int_{\Omega} \frac{\langle x-y, n(y)\rangle}{4\pi||x-y||^3} u(y)\mathrm{d}y, x \in \Omega \tag{6}$$

where Ω is the domain surface. Equation(5) and (6) correspond to Eq.(1) and the details of them are described in [3]. The spheres were set at a distance of 0.25 m from the ground with zero electric potential. The radius of the spheres was 0.25 m, and the electric potential of the spheres was 1 V.

For the numerical evaluations, we used the BDW and the KNL processors, which have a 256-bit SIMD unit and a 512-bit SIMD unit, respectively. The processor specifications are summarized in Table 1. For both processors, Intel Fortran compiler ver. 18.0.1 was used. The compiler options for BDW were `-align array64byte -xAVX2 -qopenmp -O3 -fpp -ipo -lm -qopt-report=5 -DSIMDLENGTH=4`, and those for KNL were `-align array64byte -xMIC-AVX512 -qopenmp -O3 -fpp -ipo -lm -qopt-report=5 -DSIMDLENGTH=8`.

4.2 Hand Tuning Using OpenMP SIMD Directives

To test the compiler vectorizations, we refactored and evaluated two user-defined functions. Vectorization with compiler directives often requires users to converse with the compiler. We tried to vectorize the user-defined functions by preparing the following series of implementations.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_46

```
real(8) function ppohBEM_matrix_element_ij_dummy(i,j,nond,nofc,nond_on_fc
    ,np,intpara_fc,nint_para_fc,dble_para_fc,ndble_para_fc,face2node)
  implicit none
  type :: coordinate
    real(8) :: x,y,z
  end type coordinate
  integer ,intent(in) :: i,j,nond,nofc,nond_on_fc,nint_para_fc,
      ndble_para_fc
  type(coordinate),intent(in) :: np(*)
  integer, intent(in) :: face2node(3,*),int_para_fc(nint_para_fc,*)
  real(8), intent(in) :: dble_para_fc(ndble_para_fc,*)
  integer :: ii,jj,j_st,j_en,lhp,ltp
  real(8) :: ans
#include "declaration.inc"
#include "call_set_args_i.inc"
#include "call_set_args_j.inc"
#include "call_set_args.inc"
#include "vectorize_func.inc"
  ppohBEM_matrix_element_ij_dummy = ans

end function ppohBEM_matrix_element_ij_dummy
```

Fig. 9. Dummy function of user-defined function. Although the function is not used in the framework, users are required to implement this function correctly.

Table 1. Processor Specifications

	Processor Name	Number of cores	Peak performance	Length of SIMD unit
BDW	Intel Xeon E5-2695 v4	18	605 GFlops	256 bit
KNL	Intel Xeon Phi 7250	68	3,046 GFlops	512 bit

H1: Original implementation without compiler directives.
H2: `!$omp simd` directives are inserted above the SIMD target loops of H1.
H3: `!$omp declare simd` directives are inserted in the function shown in Fig. 3 and all user-defined functions called from the function of H2 shown in Fig.3.
H4: A `simdlen(SIMDLENGTH)` clause is attached to each `!$omp simd` and `!$omp declare simd` directive of H3.
H5: Replace the user-defined functions of H4 with the `set_args` and `vectorize_func` interfaces.
H6: The interfaces `set_args_i` and `set_args_j` are used as alternatives to `set_args` of H5.
H7: `linear` clauses are attached to a `!$omp declare simd` directive of `vectorize_func` of H6.
H8: `uniform` clauses are used as constant variables instead of `linear` clauses of H7.

Implementations H1-H4 are based on the original framework. The differences among these implementations are only in terms of the OpenMP directives. Therefore, users familiar with SIMD can implement H1-H4 with relative ease.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_46

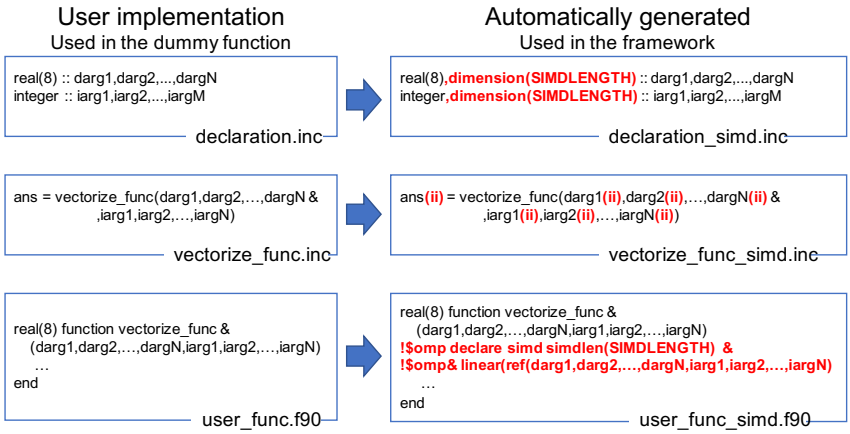

Fig. 10. The users program automatically transformed at the compile time.

Implementations H5-H8 are based on the proposed framework. Specifically, implementation H7 corresponds to the automatically generated program. Note that implementation H8 is more optimized than implementation H7. However, to automatically generate implementation H8, syntactic analysis is required. This will be realized in the future.

Figures 11-14 show the increase in speed compared to the speed of implementation H1, and Table 2 summarizes the elapsed times of implementations H1 and H7. The results discussed in this section are the averages of 10 measurements. As summarized in Table 2, although we recommend the BEM-BB H-matrix version, we evaluated the dense matrix version, the performance of which depends to a greater extent on the user-defined function. The main difference between the two functions from the viewpoint of SIMD vectorization is whether the function has a branch. Although the increase in speed in case of the dielectric problem shows a trend similar to that in case of the perfect conductor problem, it is slightly worse owing to the branch divergence caused by the dielectric function. The results obtained by solving the perfect conductor problem on a machine with the KNL processor (Fig.11) show that the proposed implementation (H7) achieved performance improvements of 4.34x and 6.62x compared to implementation H0 for the $\mathcal{H}$-matrix and the dense matrix versions, respectively. The theoretical speedup with SIMD vectorization equals `SIMDLENGTH`, and the results of the dense matrix version demonstrate that the framework improves SIMD vectorization performance considerably. In the results obtained on a machine with the BDW processor (Fig.13), implementation H7 achieved performance improvements of 2.22x and 2.44x compared to implementation H0 for the $\mathcal{H}$- matrix and the dense matrix versions, respectively

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_46

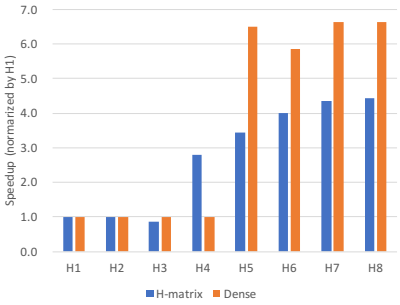

Fig. 11. Solving perfect conductor problem using KNL processor

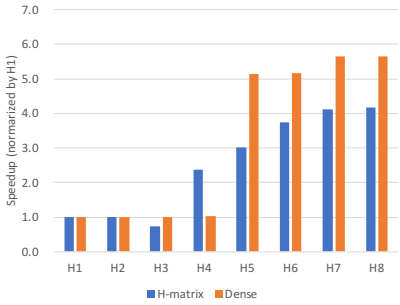

Fig. 12. Solving dielectric problem using KNL proccesor

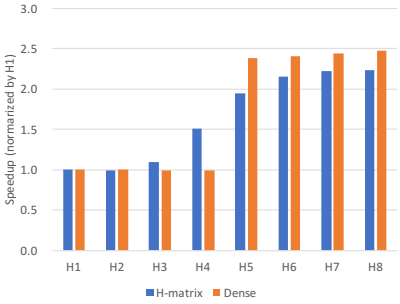

Fig. 13. Solving perfect conductor problem using BDW processor

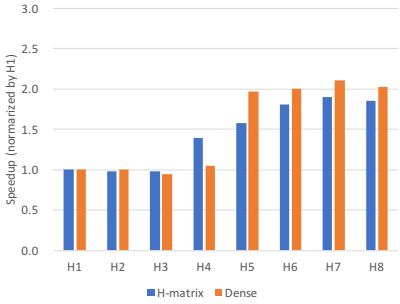

Fig. 14. Solving dielectric problem using BDW processor

5 Related Work

The literature contains many studies about software frameworks for parallel PDE solvers of the finite element method, such as GeoFEM [10] and Free FEM++ [11]. Moreover, $\mathcal{H}$-matrices have been used in a few BEM applications [8, 12, 13], and parallelized in their application. Although many frameworks allow for MPI + OpenMP hybrid parallelization, few frameworks support SIMD vectorization, which highly depends on user-defined functions. The main contribution of this study is a SPMD-like SIMD vectorization method that handles data access and computation separately, and hides SIMD-related aspects in the framework. The method uses the characteristics of BEM analysis: the kernel function is relatively computationally intensive, and there exists no data dependency among the calculations of elements of coefficient matrix.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_46

Table 2. The elapsed times of coefficient generation component of original implementation (H1) and implementation of proposed framework (H7)

	Perfect conductor				Dielectric			
	KNL		BDW		KNL		BDW	
	H-matrix	Dense	H-matrix	Dense	H-matrix	Dense	H-matrix	Dense
H1	10.00	215.0	10.51	233.2	13.07	249.5	13.53	265.5
H7	2.307	32.47	4.728	95.61	3.167	44.11	7.140	126.10

6 Conclusion

We refined the open-source framework for parallel BEM analysis to enhance SIMD vectorizations, which is important for realizing high-performance computing. By using the refined framework design, we could successfully separate SIMD-related aspects from the user-defined function, which depends on target applications. We evaluated the proposed framework by solving two static electric field analysis problems containing different user-defined functions on a BDW processor and a KNL processor. The numerical results demonstrated the improved performance of the framework. Specifically, in solving the perfect conductor problem by using the KNL processor, we achieved performance improvements of 4.34x and 6.62x in the $\mathcal{H}$-matrix case and the dense matrix cases, respectively.

The main contribution of this paper is separating the SIMD-related aspects from the user-defined function and hiding them to minimize the difficulties associated with SIMD. This SPMD-like SIMD vectorization technique can be used for other applications. In the proposed framework, the arguments of the `vectorize_func` must be scalar variable. This specification is not user-friendly but compiler-friendly. For example, to adjust the user-defined functions in the proposed framework, we separated the vector argument `coordinate(3)` to scalars `x`, `y`, and `z`. This type of transformation is a typical Array of Structure (AoS) to Structure of Array (SoA) transformation. To improve the not user-friendly specification, we will challenge to support the AoS to SoA transformation in future.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Number 16H06679 and 17H01749.

References

1. W. Hackbusch. A sparse matrix arithmetic based on h-matrices. part i: Introduction to h-matrices. *Computing*, 62(2):89–108, Apr 1999.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_46

2. W. Hackbusch and B. N. Khoromskij. A sparse h -matrix arithmetic. part ii: Application to multi-dimensional problems. *Computing*, 64(1):21–47, January 2000.
3. Steffen Börm ; Lars Grasedyck ; Wolfgang Hackbusch. Hierarchical matrices. Technical report, Max Planck Institute for Mathematics in the Sciences, 2003.
4. Steffen Börm and Joana Bendoraityte. Distributed h^2-matrices for non-local operators. *Computing and Visualization in Science*, 11(4):237–249, Sep 2008.
5. Rio Yokota, L.A. Barba, Tetsu Narumi, and Kenji Yasuoka. Petascale turbulence simulation using a highly parallel fast multipole method on gpus. *Computer Physics Communications*, 184(3):445 – 455, 2013.
6. ppOpen-HPC. Open Source Infrastructure for Development and Execution of Large-Scale Scientific Applications on Post-Peta-Scale Supercomputers with Automatic Tuning (AT), `http://ppopenhpc.cc.u-tokyo.ac.jp/ppopenhpc/`.
7. Akihiro Ida, Takeshi Iwashita, Takeshi Mifune, and Yasuhito Takahashi. Parallel hierarchical matrices with adaptive cross approximation on symmetric multiprocessing clusters. *Journal of Information Processing*, 22(4):642–650, 2014.
8. Takeshi Iwashita, Akihiro Ida, Takeshi Mifune, and Yasuhito Takahashi. Software framework for parallel bem analyses with h-matrices using mpi and openmp. *Procedia Computer Science*, 108:2200 – 2209, 2017.
9. S. Kurz, O. Rain, and S. Rjasanow. The adaptive cross-approximation technique for the 3d boundary-element method. *IEEE Transactions on Magnetics*, 38(2):421–424, Mar 2002.
10. Hiroshi Okuda, Kengo Nakajima, Mikio Iizuka, Li Chen, and Hisashi Nakamura. Parallel finite element analysis platform for the earth simulator: Geofem. In Peter M. A. Sloot, David Abramson, Alexander V. Bogdanov, Yuriy E. Gorbachev, Jack J. Dongarra, and Albert Y. Zomaya, editors, *Computational Science — ICCS 2003*, pages 773–780, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.
11. F. Hecht. New development in freefem++. *J. Numer. Math.*, 20(3-4):251–265, 2012.
12. S. Kurz, O. Rain, and S. Rjasanow. The adaptive cross-approximation technique for the 3d boundary-element method. *IEEE Transactions on Magnetics*, 38(2):421–424, Mar 2002.
13. Makiko Ohtani, Kazuro Hirahara, Yasuto Takahashi, Takane Hori, Mamoru Hyodo, Hiroshi Nakashima, and Takeshi Iwashita. Fast computation of quasi-dynamic earthquake cycle simulation with hierarchical matrices. *Procedia Computer Science*, 4:1456–1465, 2011.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_46