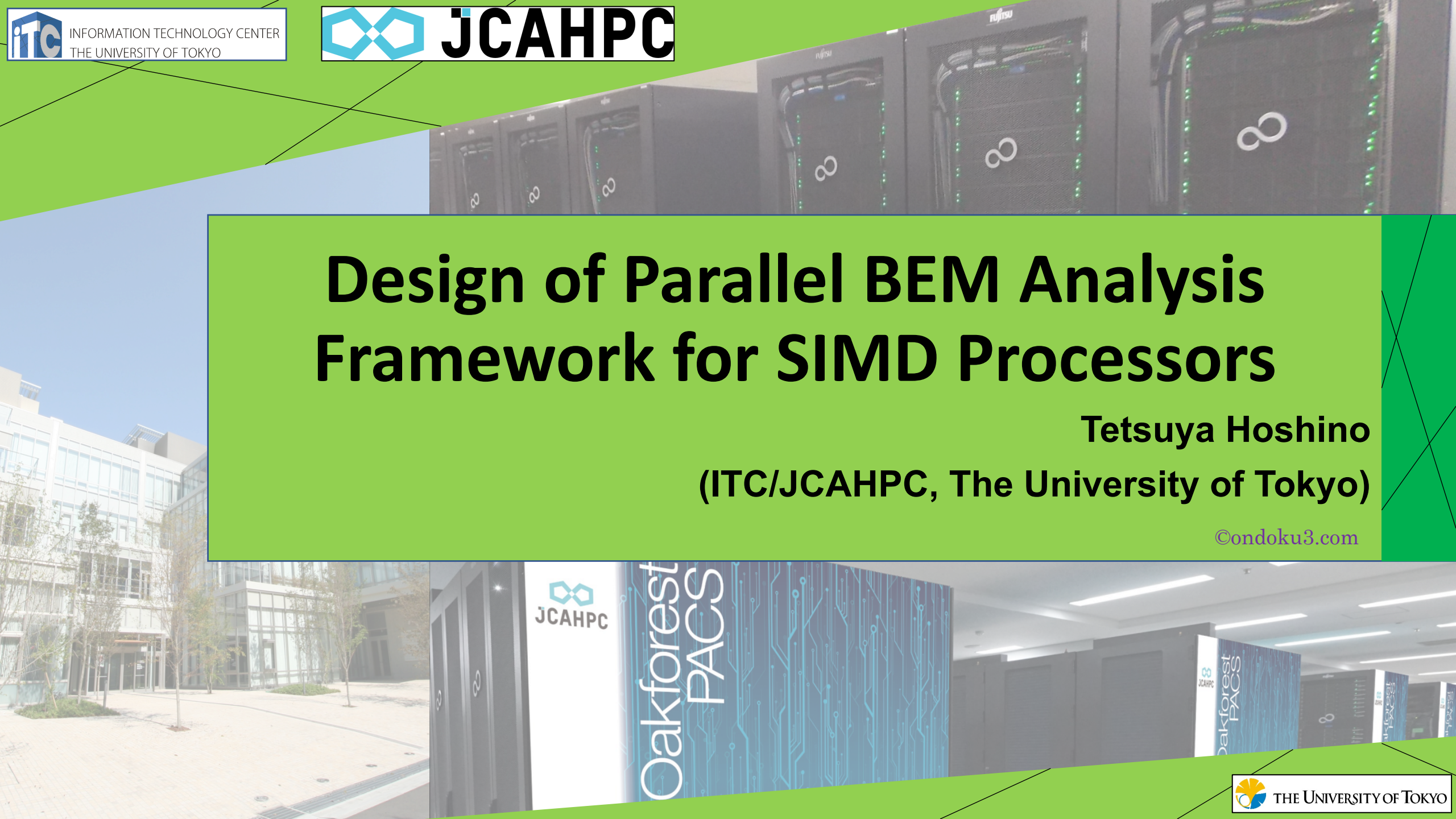
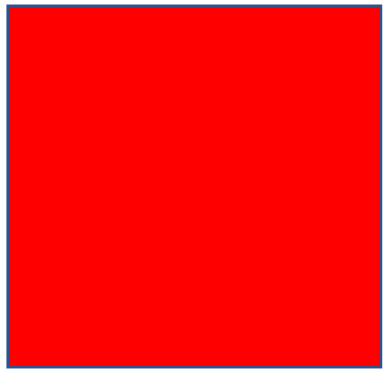

INFORMATION TECHNOLOGY CENTER
THE UNIVERSITY OF TOKYO
JCAHPC
Design of Parallel BEM Analysis Framework for SIMD Processors
Tetsuya Hoshino
(ITC/JCAHPC, The University of Tokyo)
©ondoku3.com
JCAHPC
Oakforest PACS
THE UNIVERSITY OF TOKYO

BEM-BB: Parallel Framework for Boundary Element Method Analyses

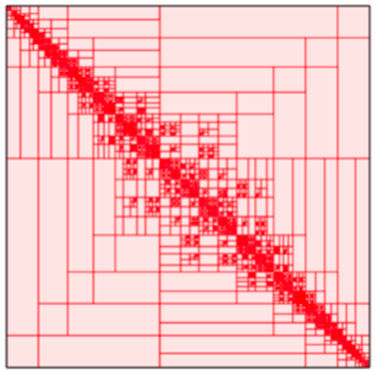
- BEM-BB (http://ppopenhpc.cc.u-tokyo.ac.jp/ppopenhpc/downloads/)
  - Fortran90 based framework
  - Provides parallel generation of coefficient matrix and parallel linear solvers
  - Parallelized with MPI + OpenMP
  - Supports dense matrices and hierarchical matrices ($\mathcal{H}$-matrices)

Dense matrix



$O(N^2)$

$\mathcal{H}$-matrix

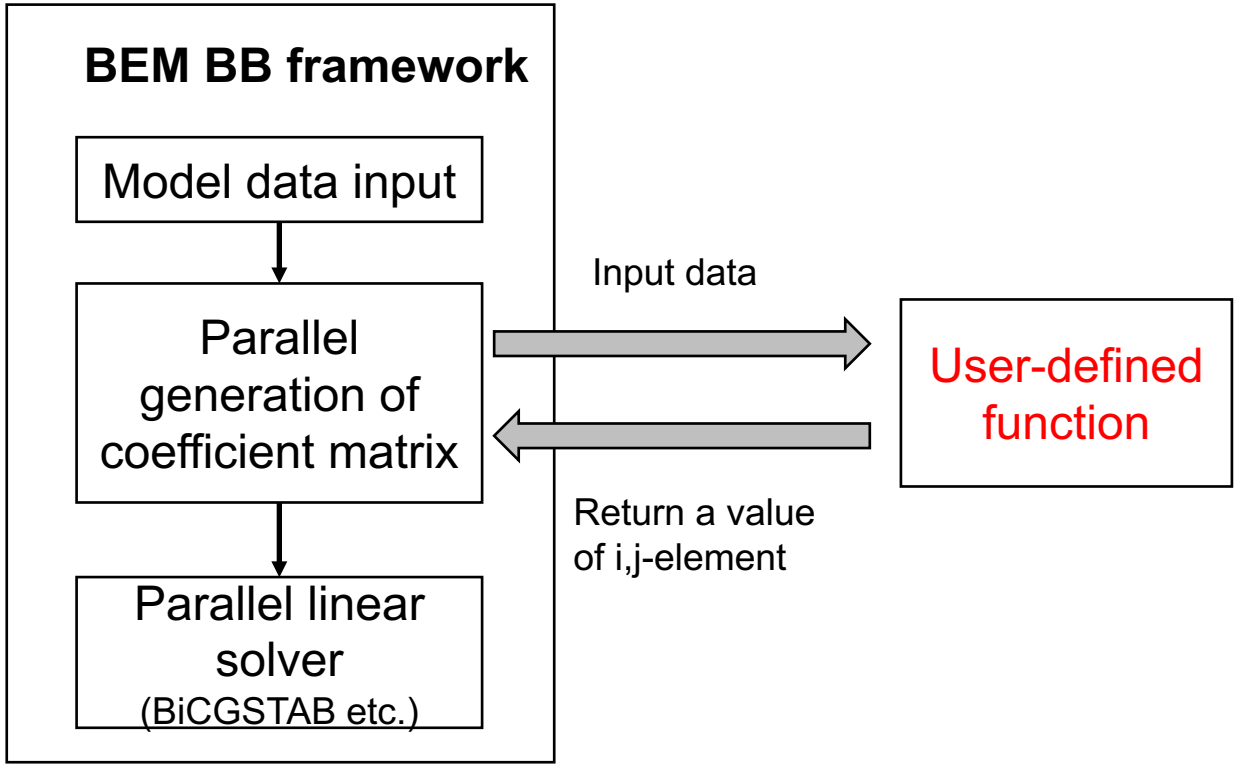


$O(N\log N)$

Framework Design of BEM-BB

Target equation of BEM analysis:

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



Framework Design of BEM-BB

Target equation of BEM analysis:

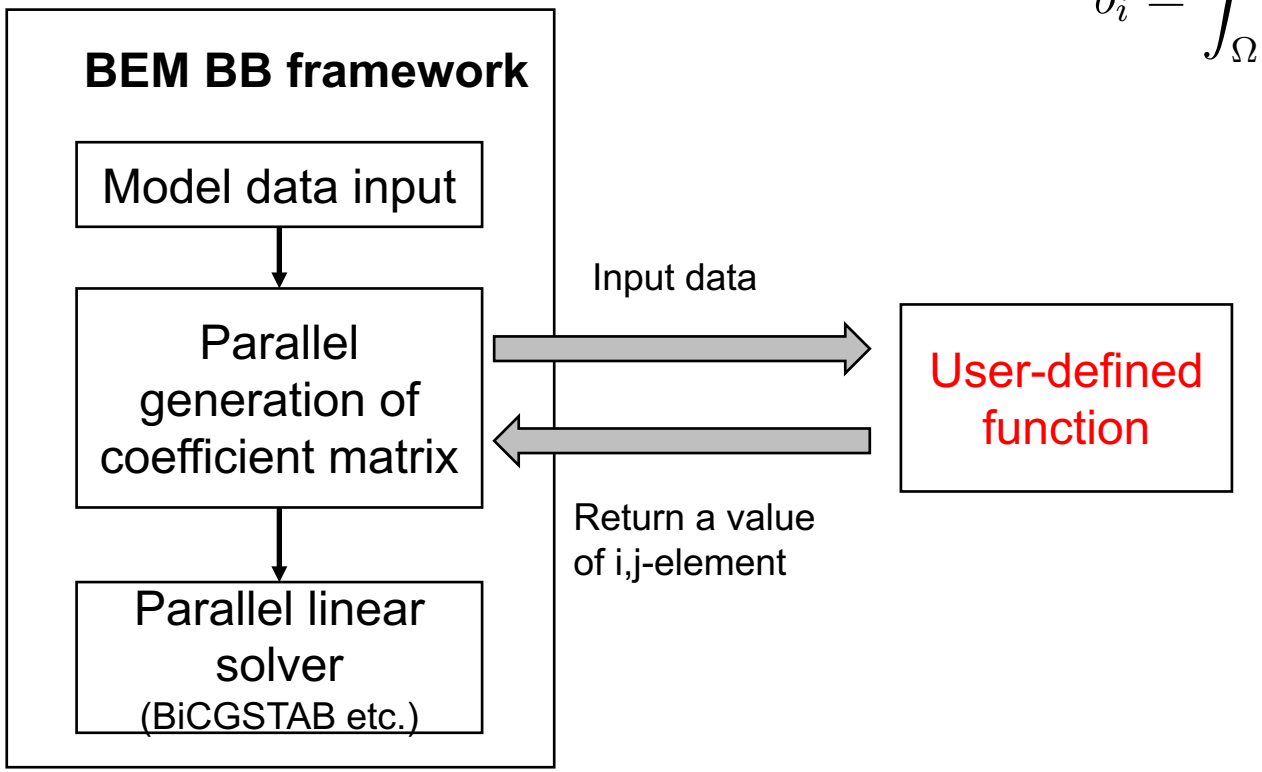
$$\int_\Omega g(x,y)u(y)\mathrm{d}y = f$$

discretization

$$A\phi = b$$

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

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



Framework Design of BEM-BB

Target equation of BEM analysis:

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

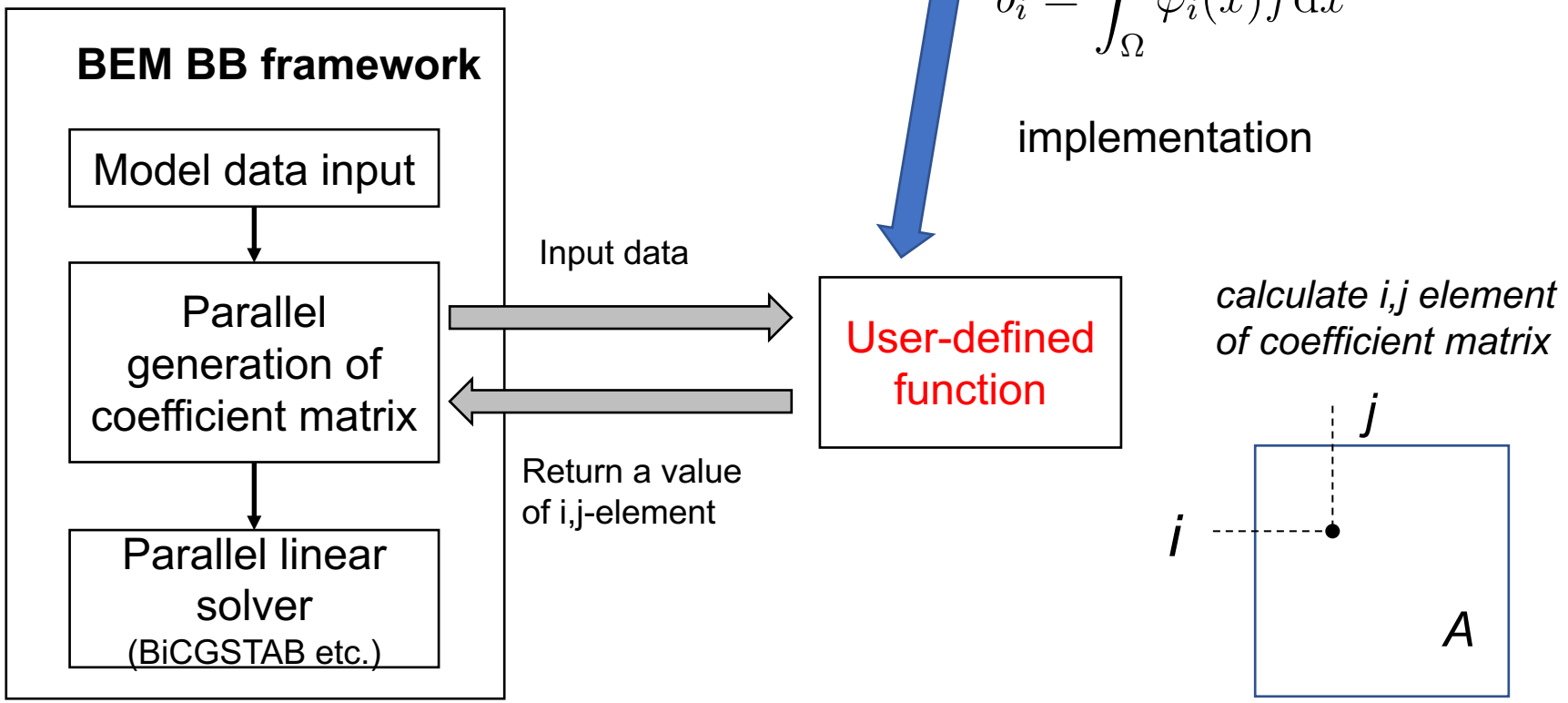
discretization

$$A\phi = b$$

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

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

implementation



calculate i,j element of coefficient matrix

Basic Idea for Vectorization

Coefficient matrix generation

```
!$omp parallel do
    do j = 1, N
      !$omp simd
      do i = 1, N
        a(i,j) = user_func(i,j,st_bemv)
      end do
    end do
!$omp end do
```

st_bemv is a structure including input model data

Basic Idea for Vectorization

Coefficient matrix generation

```
!$omp parallel do
    do j = 1, N
      !$omp simd
      do i = 1, N
        a(i,j) = user_func(i,j,st_bemv)
      end do
    end do
!$omp end do
```

This SIMD directive does not work!

st_bemv is a structure including input model data

Basic Idea for Vectorization

Coefficient matrix generation

```
!$omp parallel do
   do j = 1, N
    !$omp simd
    do i = 1, N
      a(i,j) = user_func(i,j,st_bemv)
    end do
   end do
!$omp end do
```

This SIMD directive does not work!

st_bemv is a structure including input model data

```
  real(8),dimension(SIMDLENGTH) :: ans
  real(8),dimension(SIMDLENGTH) :: arg1,arg2,...
!$omp parallel do
  do j = 1, N
    do i = 1, N, SIMDLENGTH
      ii = 1
      do jj = i, min(i+SIMDLENGTH-1, N)
        call set_args(i,j,st_bemv,arg1(ii),arg2(ii),…)
        ii = ii+1
      end do
      !$omp simd
      do ii = 1, SIMDLENGTH
        ans(ii) = vectorize_func(arg1(ii),arg2(ii),…)
      end do
      ii = 1
      do jj=i,min(i+SIMDLENGTH-1, N)
        a(i,j) = ans(ii)
        ii = ii+1
      end do
    end do
  end do
!$omp end parallel
```

This loop is sequentially executed

This loop is obviously vectorizable

Basic Idea for Vectorization

Structure including input model data

User function

Original

```
real(8) function user_func(i,j,st_bemv)
  integer :: i,j
  type(BemInput) :: st_bemv
  real(8) :: a1, a2, …

  a1 = st_bemv%a1(i,j)
  a2 = st_bemv%a2(i,j)
  …
  ! calculate i,j value of coefficient

end function user_func
```

User implementation

Data access

```
subroutine set_args(i,j,st_bemv, a1, a2, …)
  integer :: i,j
  type(hacapkInput) :: st_bemv
  real(8) :: a1, a2, …
  a1 = st_bemv%a1(i,j)
  a2 = st_bemv%a2(i,j)
  …
end subroutine set_args
```

Computation

```
real(8) function vectorize_func(a1, a2, …)
!$omp declare simd(vectorize_func)        &
!$omp simdlen(SIMDLENGTH)                 &
!$omp linear(ref(a1, a2, …))
  real(8) :: a1, a2, …
 ! calculate i,j value of coefficient
end function vectorize_func
```

Fill-in-the-blank puzzle-like user interface

```
real(8) function user_func_dummy(i,j,st_bemv)
  implicit none
  integer ,intent(in) :: i,j
  type(BemInput) :: st_bemv
  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"
  user_func_dummy = ans
end function user_func_dummy
```

1. Implement include files
2. Implement “set_args” and “vectorize_func” in “user func.f90”.
3. Correctly implement the dummy without modifying the dummy function itself
4. Provide SIMDLENGTH of the target processor by using the -D compiler flag

Numerical Evaluations

T. Hoshino et al. “Design of Parallel BEM Analyses Framework for SIMD Processors” (ICCS 2018)

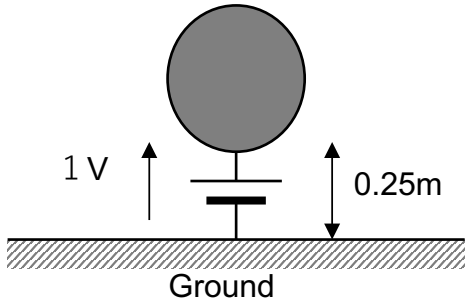
Test model of electrostatic field analysis

- Perfect conducting sphere
- Dielectric sphere
  - including branch divergence

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

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

User-defined functions depend on these integral equations

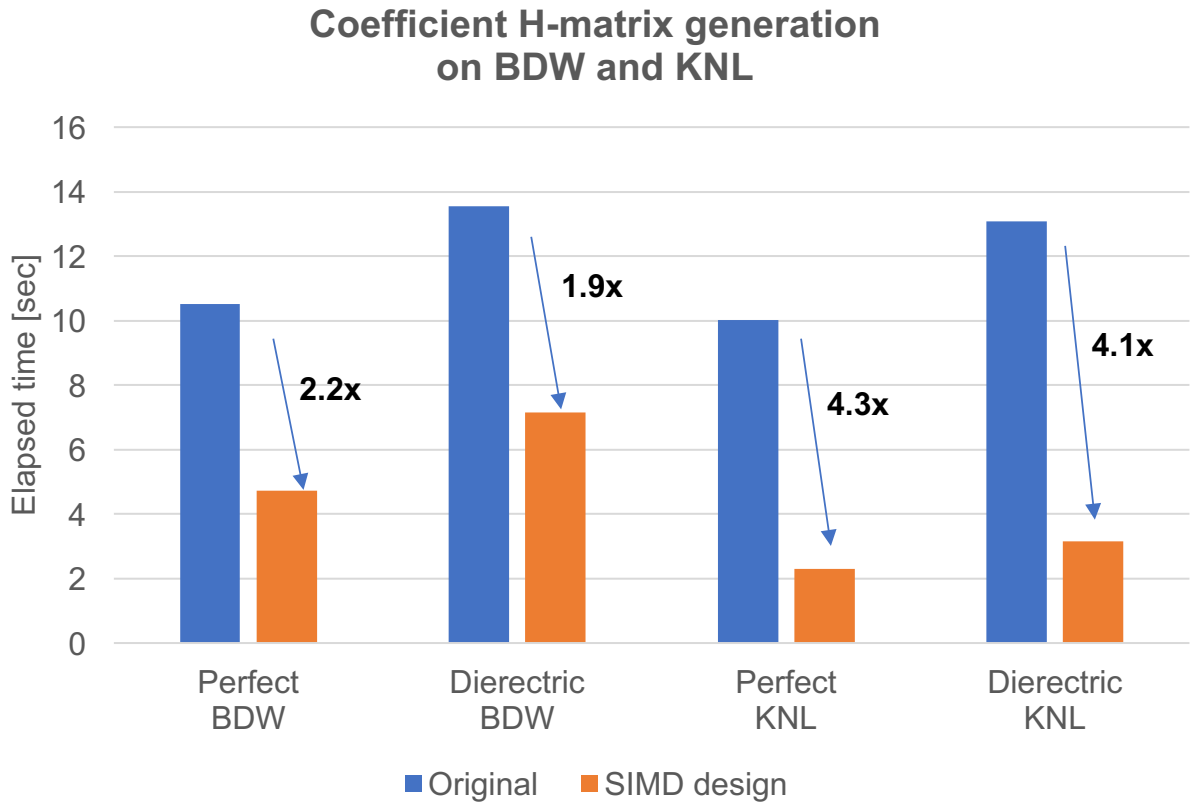


Evaluation Environments

- BDW : Intel Xeon E5-2695 v4, 18 core
- KNL : Intel Xeon Phi 7250, 68 core
- Compiler : Intel compiler 18.0.1
  - -qopenmp -O3 -ipo -align array64byte -xAVX2 (BDW) –xMIC-AVX512 (KNL)

Performance comparison

- BDW : approximately **2x** speedup
- KNL : over **4x** speedup
  - In the case of dense matrix generation, new design achieved at most **6.6x** speedup



Conclusion

- We propose new framework design of BEM-BB for SIMD processors
  - The SIMD vectorization strategy can be used in other compute bounded applications
- We evaluate the proposed framework on BDW and KNL with electrostatic field analysis
  - BDW: **2.22x** and **2.44x** speedup for H-matrix and Dense-matrix construction, respectively
  - KNL: **4.34x** and **6.62x** speedup for H-matrix and Dense-matrix construction, respectively
- For more details…
  - T. Hoshino et al. “Design of Parallel BEM Analyses Framework for SIMD Processors” (ICCS 2018)

Thank you for watching