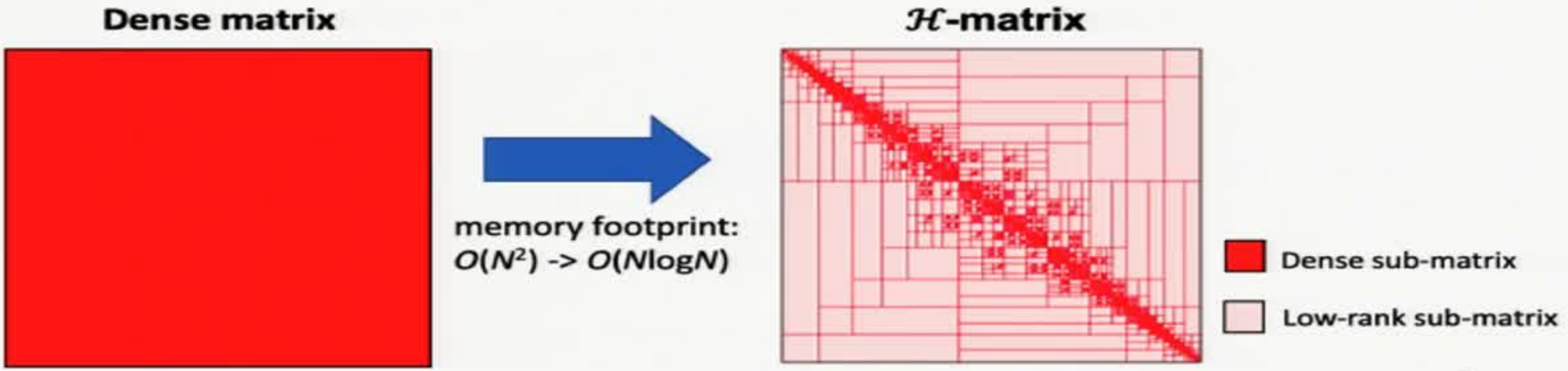
What is $\mathcal{H}$-matrix?

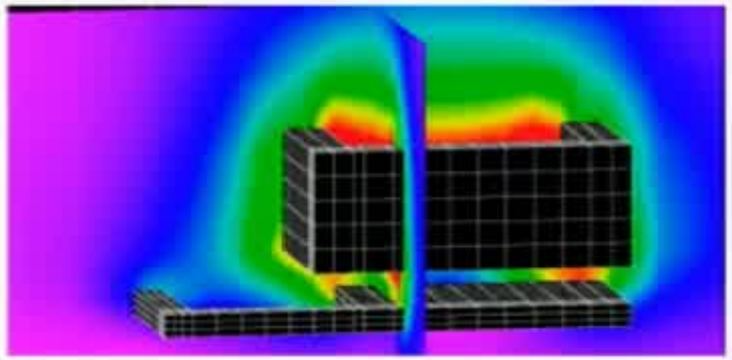
- $\mathcal{H}$-matrices are an approximation technique for dense matrices
  - ✓ Such as the coefficient matrix of the boundary element method (BEM)
  - ✓ Expressed by a set of low-rank approximated and small dense sub-matrices
- $\mathcal{H}$-matrices reduce the memory footprint, computational cost
  - ✓ For N unknowns,
    - Dense matrix: $O(N^2)$ memory and over $O(N^2)$ computational cost
    - **$\mathcal{H}$-matrix**: $O(N\log N)$ memory and over $O(N\log N)$ computational cost



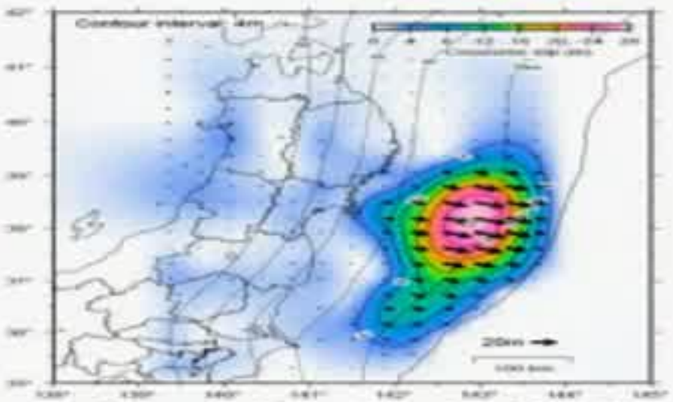
$\mathcal{H}$ACApK

- $\mathcal{H}$ACApK (http://ppopenhpc.cc.u-tokyo.ac.jp/ppopenhpc/downloads/)
  - ✓Fortran90 based library for Hierarchical matrices ($\mathcal{H}$-matrices)
  - ✓Parallelized with MPI + OpenMP in original
  - ✓Provides parallel generation of $\mathcal{H}$-matrix and parallel linear solvers
  - ✓**OpenACC + CUDA for GPU**
  - ✓**SIMD for Intel KNL and Broadwell Processors**
- Target applications

・ Electromagnetic fields

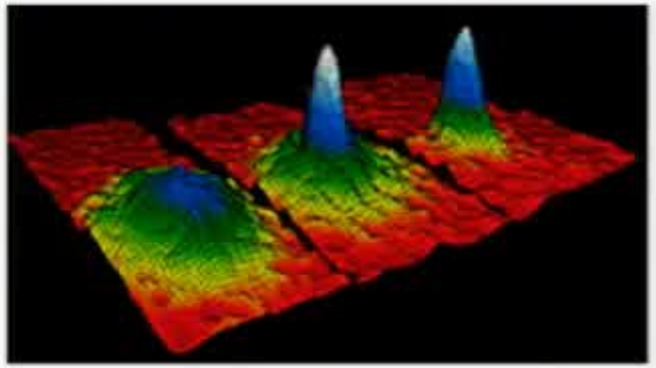


・ Earthquake cycle



・ Transcribed from **Ohtani at al (2011)**

・ Quantum mechanics



・ Transcribed from **NIST Image Gallery**

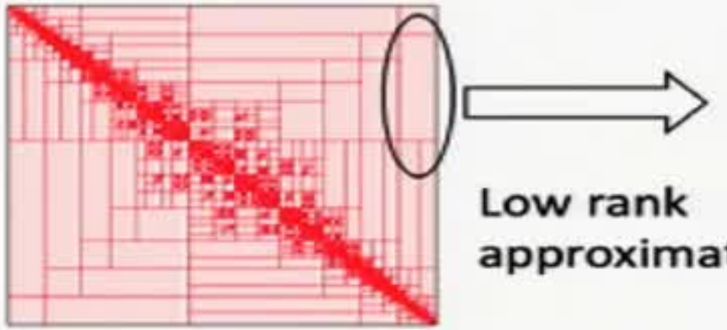
Hierarchical matrices

Target Example

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

singular kernel : $g(x,y) \in \mathrm{span}(\{|x-y|^{-p}, p>0\})$

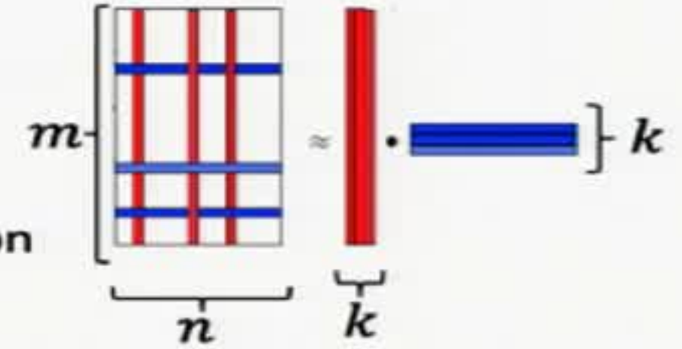
Hierarchical matrices



Full-Rank

Low-Rank

Low rank approximation



$$O(mn) \Rightarrow O(k(m+n))$$

- # of matrix elements

$\mathcal{H}$ACApK component

Target equation of BEM analysis:

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

discretization

$$A\phi = b$$

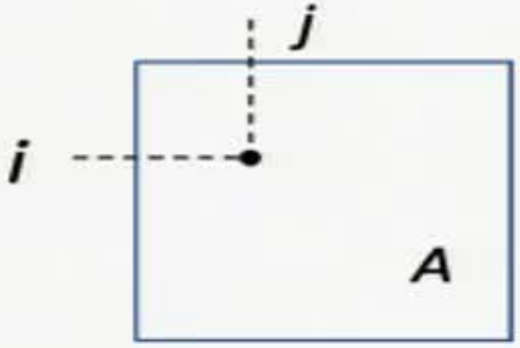
$$A_{ij} = \int_\Omega \varphi_i(x) \int_\Omega g(x,y)\varphi(y)\mathrm{d}y\mathrm{d}x$$

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

prepare as a fortran function

User-defined function

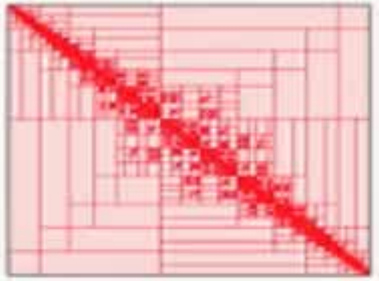
calculate i,j element of coefficient matrix

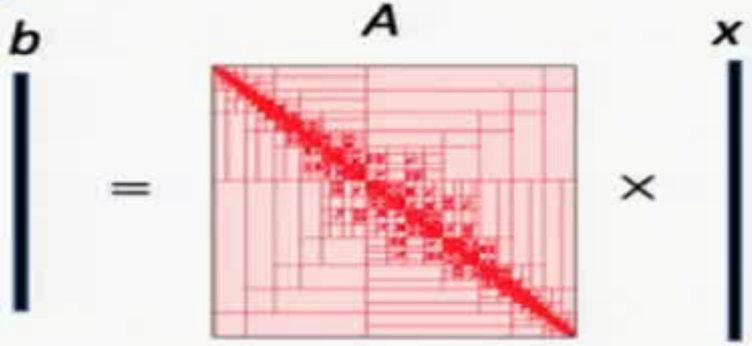


$\mathcal{H}$ACApK

$\mathcal{H}$-matrix construction

$\mathcal{H}$-matrix linear solver (BiCGSTAB etc.)





2 problems of $\mathcal{H}$ACApK for many-core processors

1. SIMD vectorization
   - ✓The user function is sequential fortran90 program without intrinsics
   - ✓The user function is black box for HACApK

User program

```
...
call Hmatrix_construction( input_data_structure )
...
```

User program calls HACApK API

HACApK API

```
subroutine Hmatrix_construction( input_data_structure )
...
H(i,j) = user_defined_function( input_data_structure )
...
end subroutine Hmatrix_construction
```

HACApK API uses user-defined func

User defined function

```
real(8) user_defined_function( input_data_structure)
...
```

How can we vectorize user-defined function ?

2 problems of $\mathcal{H}$ACApK for many-core processors

2. Load balancing

✓The $\mathcal{H}$-matrix structure depends on data set

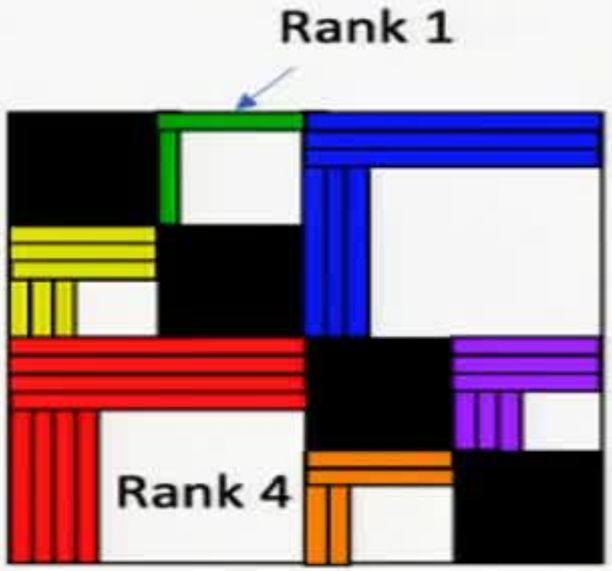
- Large sub-matrices can be appeared

Input data set

Output $\mathcal{H}$-matrix

✓The rank of each sub-matrix is variable and unknown before $\mathcal{H}$-matrix generation

- $\mathcal{H}$ACApK uses variable rank method



Computational cost of each sub-matrix is variable

Approach and contributions

- SIMD vectorization
  - ✓We provide new interface of user-defined function and auto-transformation system
    - According to the auto-transformation, users can vectorize the function with small knowledge of SIMD
    - Evaluated on Intel Broadwell and Intel Xeon Phi KNL
  - ✓For more detail, " *Design of Parallel BEM Analyses Framework for SIMD Processors* " (ICCS 2018)
- Load balancing
  - ✓We propose load-balancing-aware algorithms for $\mathcal{H}$-matrices with Adaptive Cross Approximation (ACA) for GPUs
    - Evaluated on NVIDIA Pascal GPU (P100)
  - ✓For more detail, " *Load-balancing-aware Parallel Algorithms of $\mathcal{H}$-matrices with Adaptive Cross Approximation for GPUs* " (IEEE Cluster 2018)

What is Problem for SIMD?

Pseudocode of $\mathcal{H}$-matrix generation

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

Is this SIMD directive work well?

Structure including user-defined input model data

Basic idea for well vectorization

➢User function callee

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

Structure including input model data

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
```

Basic idea for well vectorization

➢User function caller

Original HACApK

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

Proposed HACApK

```
 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

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

Auto-transformation

User implementation
Used in the dummy function

```
real(8) :: darg1,darg2,...,dargN
integer :: iarg1,iarg2,...,iargM
```

declaration.inc

```
ans = vectorize_func(darg1,darg2,...,dargN &
                    ,iarg1,iarg2,...,iargN)
```

vectorize_func.inc

```
real(8) function vectorize_func &
   (darg1,darg2,...,dargN,iarg1,iarg2,...,iargN)
   ...
end
```

user_func.f90

Automatically generated
Used in the framework

```
real(8),dimension(SIMDLENGTH) :: darg1,darg2,...,dargN
integer,dimension(SIMDLENGTH) :: iarg1,iarg2,...,iargM
```

declaration_simd.inc

```
ans(ii) = vectorize_func(darg1(ii),darg2(ii),...,dargN(ii) &
                    ,iarg1(ii),iarg2(ii),...,iargN(ii))
```

vectorize_func_simd.inc

```
real(8) function vectorize_func &
   (darg1,darg2,...,dargN,iarg1,iarg2,...,iargN)
!$omp declare simd simdlen(SIMDLENGTH) &
!$omp& linear(ref(darg1,darg2,...,dargN,iarg1,iarg2,...,iargN)
   ...
end
```

user_func_simd.f90

Basic idea for well vectorization

- Transformed include files are used in HACApK API

```
 real(8),dimension(SIMDLENGTH) :: ans
 #include "declaration_simd.inc"
!$omp parallel do
 do j = 1, N
   do i = 1, N, SIMDLENGTH
     ii = 1
     do jj = i, min(i+SIMDLENGTH-1, N)
 #include "call_set_args_simd.inc"
       ii = ii+1
     end do
     !$omp simd
     do ii = 1, SIMDLENGTH
 #include  "vectorize_func_simd.inc"
     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
```

Electrostatic Field Analysis

- Potential operator

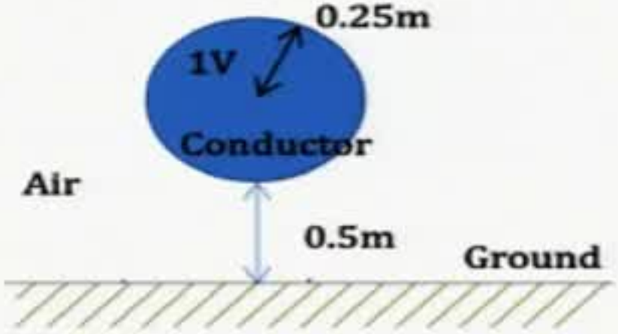
$$P[u](x) := \int_{\Omega} \frac{1}{4\pi\|x-y\|} u(y)\mathrm{d}y, x \in \Omega$$

- Operations included in User-defined function

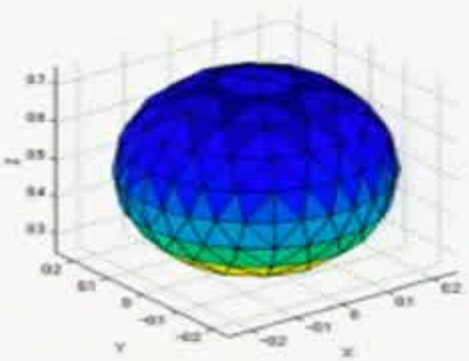
op	#
+	48
-	75
*	113
/	16
abs	2
sqrt	12
log	3
atan2	9

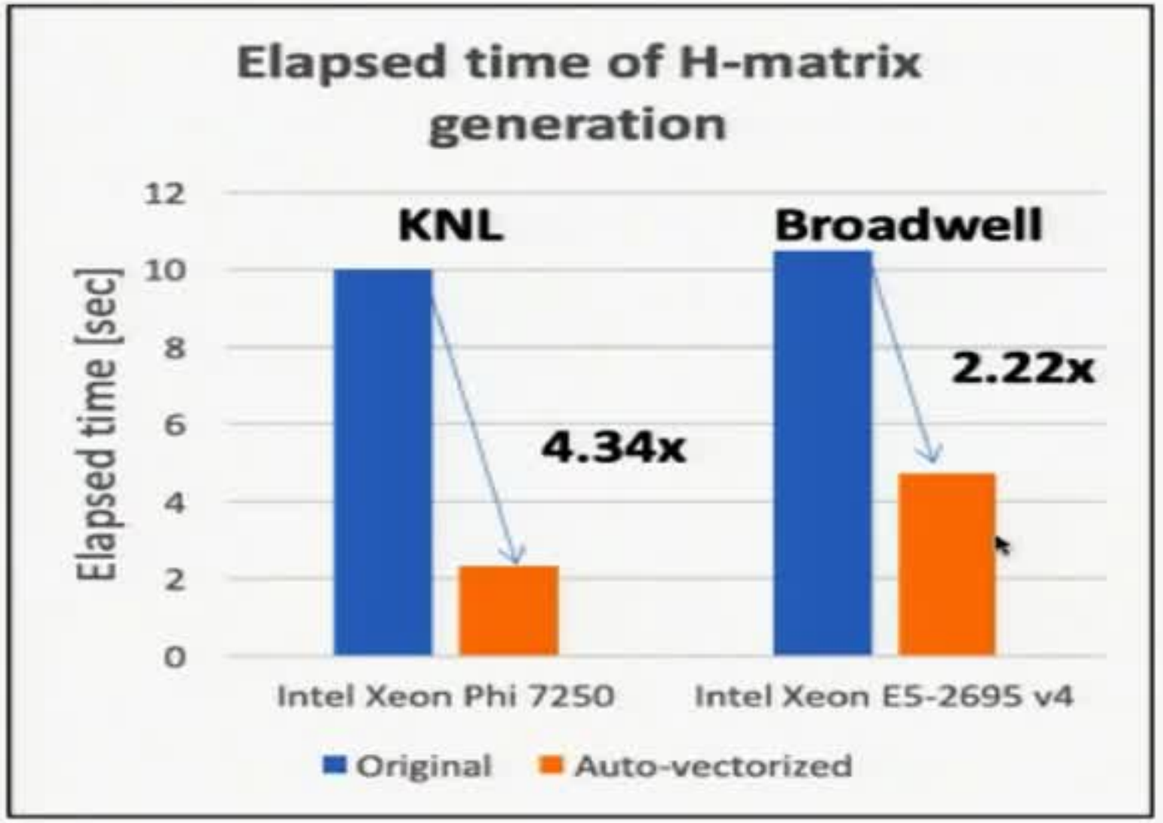
Each SIMD element executes these operations

Analysis condition



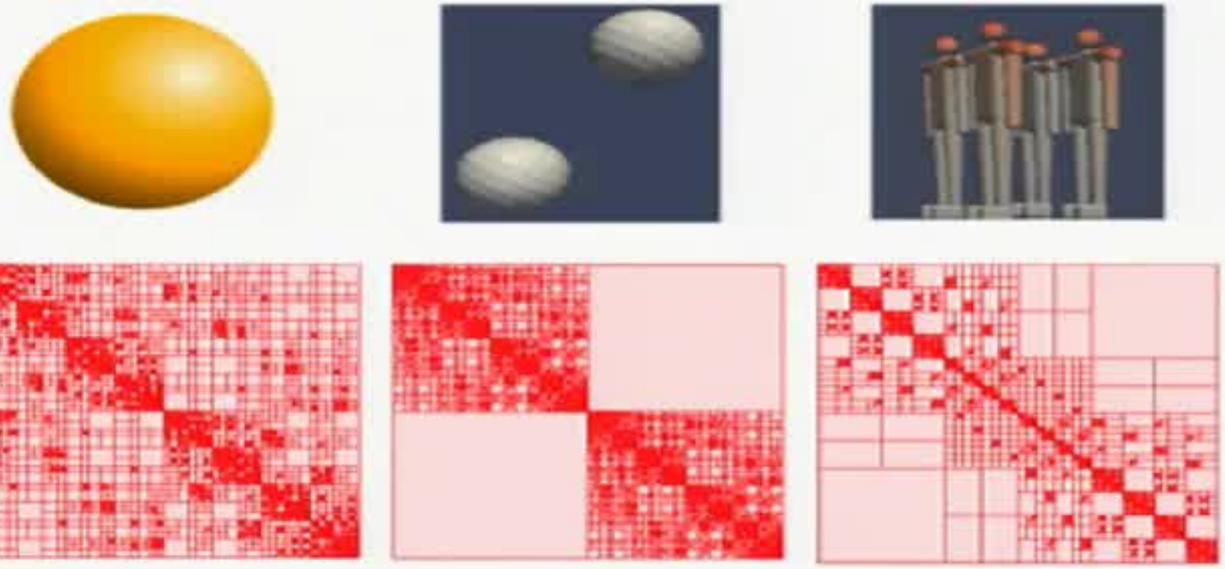
Analysis result



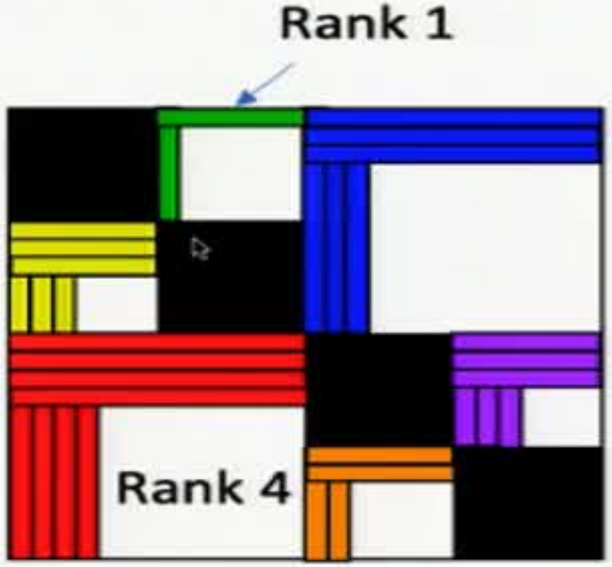


Load balancing

- The $\mathcal{H}$-matrix structure depends on data set
  - ✓ Large sub-matrices can be appeared



- The rank of each sub-matrix is variable and unknown before $\mathcal{H}$-matrix generation
  - ✓ $\mathcal{H}$ACApK uses variable rank method

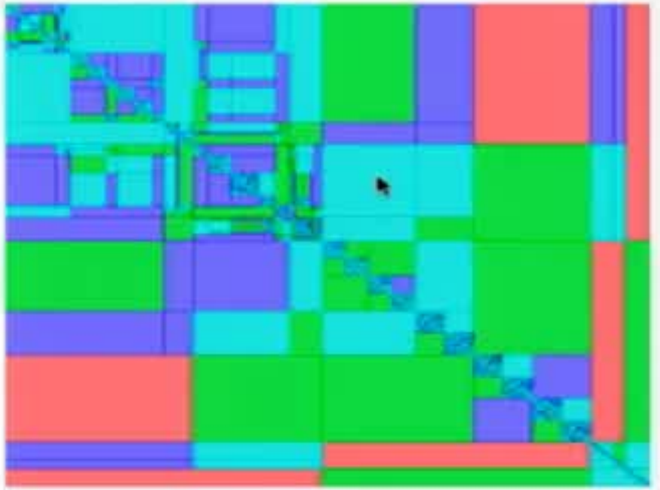


Computational cost of each sub-matrix is variable

Naïve load balancing

1. Assuming the rank is constant
2. Threads are assigned to minimize the difference of the computational cost as much as possible

Thread assignment for
4 threads



Load Balancing Efficiency of $\mathcal{H}$-matrix construction

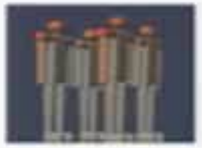
$$E := \frac{L}{\#\,\text{threads} * L_{max}}$$

L : Total computational load

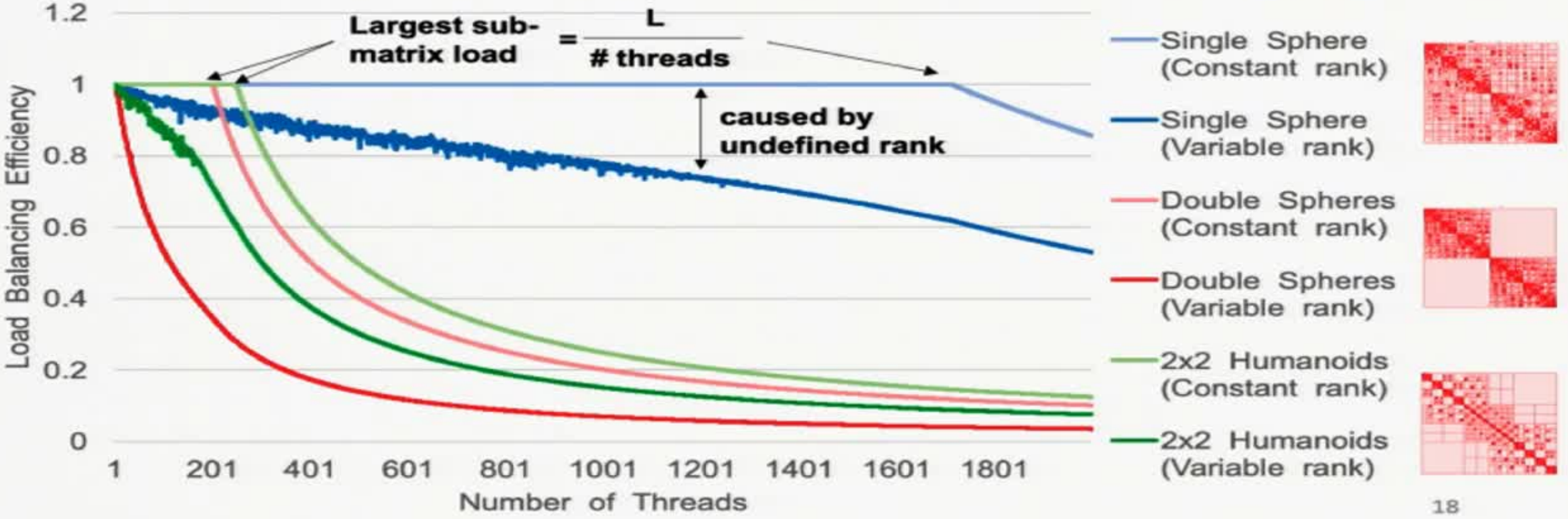
L_{max} : Max load of threads

single sphere

double spheres



2x2 humanoids



Naïve algorithm of $\mathcal{H}$-matrix construction

Sub-matrix loop executed in parallel

ACA convergence loop executed sequentially

ACA is independently applied for all sub-matrices

$\mathcal{H}$-matrix making with ACA

```
FOR ALL sub-matrices DO
choose first column vector
DO k = 1, KMAX
      FOA ALL column vector element DO
         calculate i, j element
         subtract the summation of 1,2,…,k-1 element
      ENDDO
      calculate max of k^th column
      FOA ALL row vector element DO
         calculate i, j element
         subtract the summation of 1,2,…,k-1 element
      ENDDO
      calculate error
      IF(error < eps) EXIT
      calculate max of k^th row
ENDDO
ENDDO
```

Proposed load-balancing-aware algorithm of $\mathcal{H}$-matrix construction

Sub-matrix loop executed in parallel

Loop interchange

ACA convergence loop executed sequentially

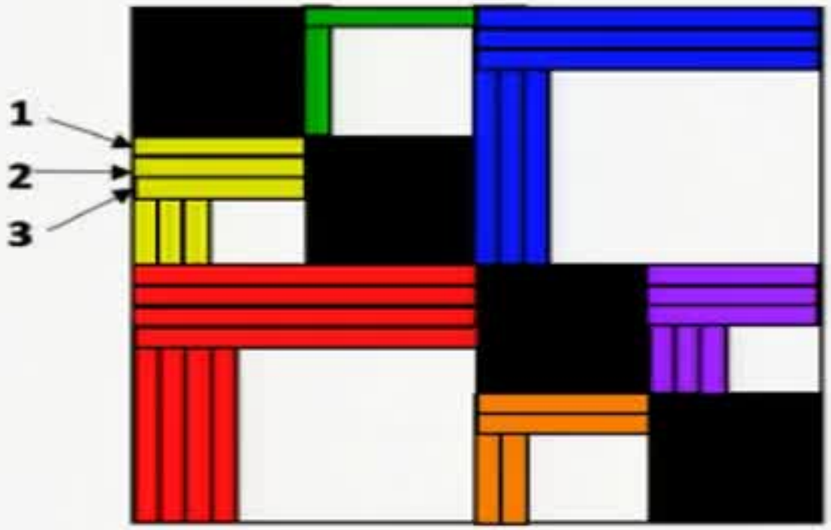
ACA is concurrently applied for all sub-matrices

$\mathcal{H}$-matrix making with ACA

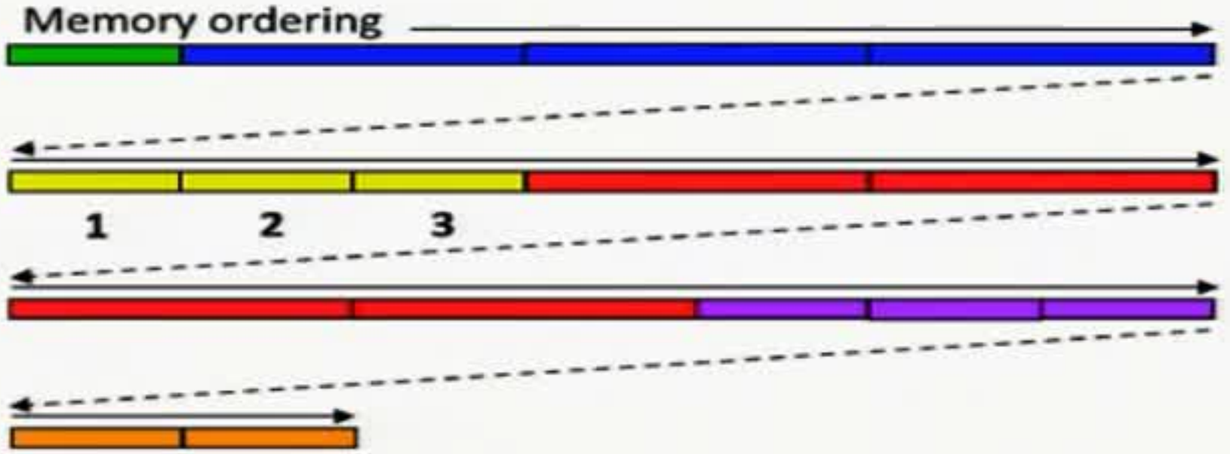
```
DO k = 1, KMAX
choose first column vector
FOR ALL sub-matrices DO
      FOA ALL column vector element DO
          calculate i, j element
          subtract the summation of 1,2,…,k-1 element
      ENDDO
      calculate max of k^th column
      FOA ALL row vector element DO
          calculate i, j element
          subtract the summation of 1,2,…,k-1 element
      ENDDO
      calculate error
      IF(error < eps) EXIT
      calculate max of k^th row
ENDDO
ENDDO
```

Naïve vs Proposed algorithm Memory Ordering

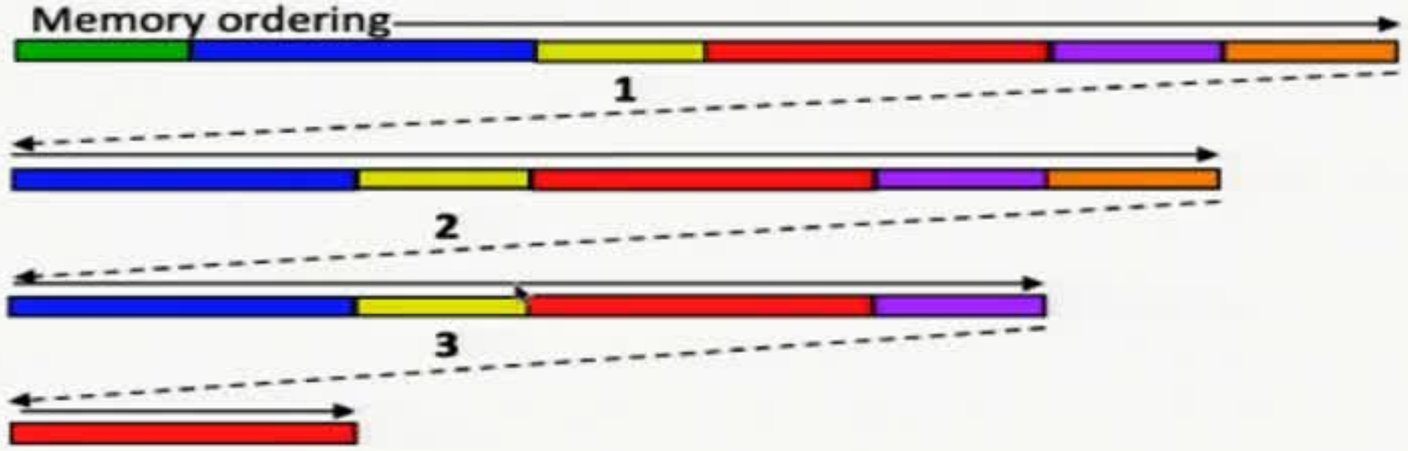
Target $\mathcal{H}$-matrix



Locality aware layout (naïve)

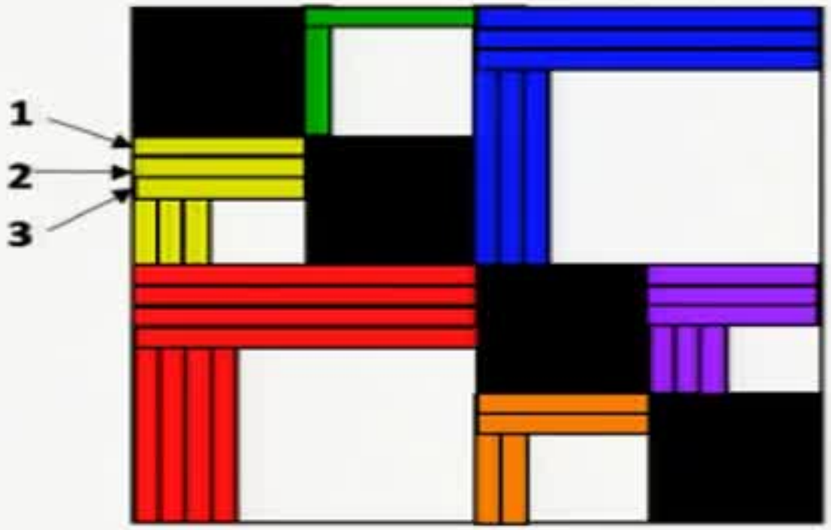


Load balancing aware layout (proposal)

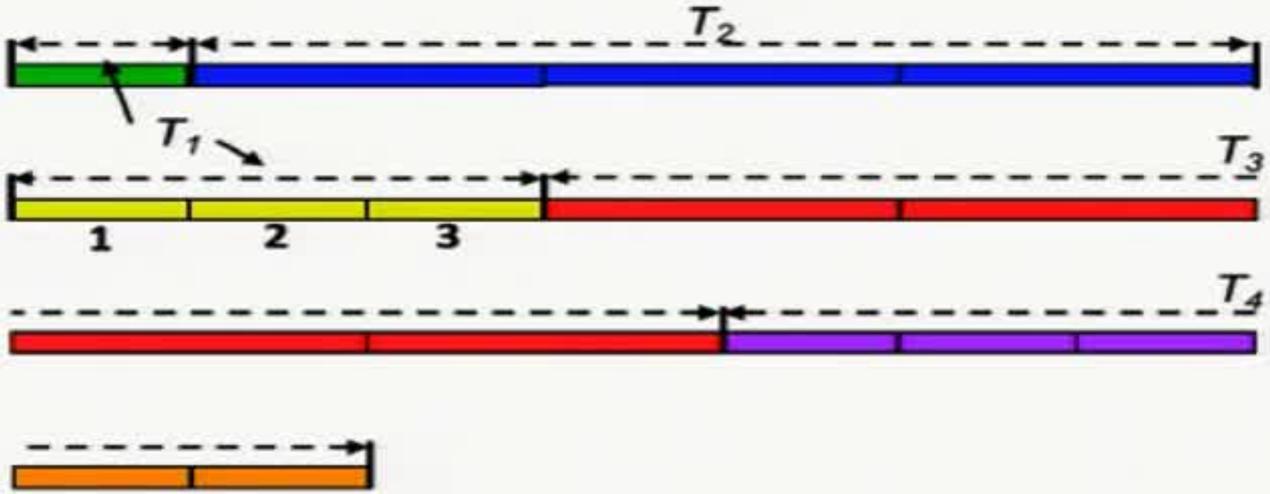


Naïve vs Proposed algorithm Parallelization

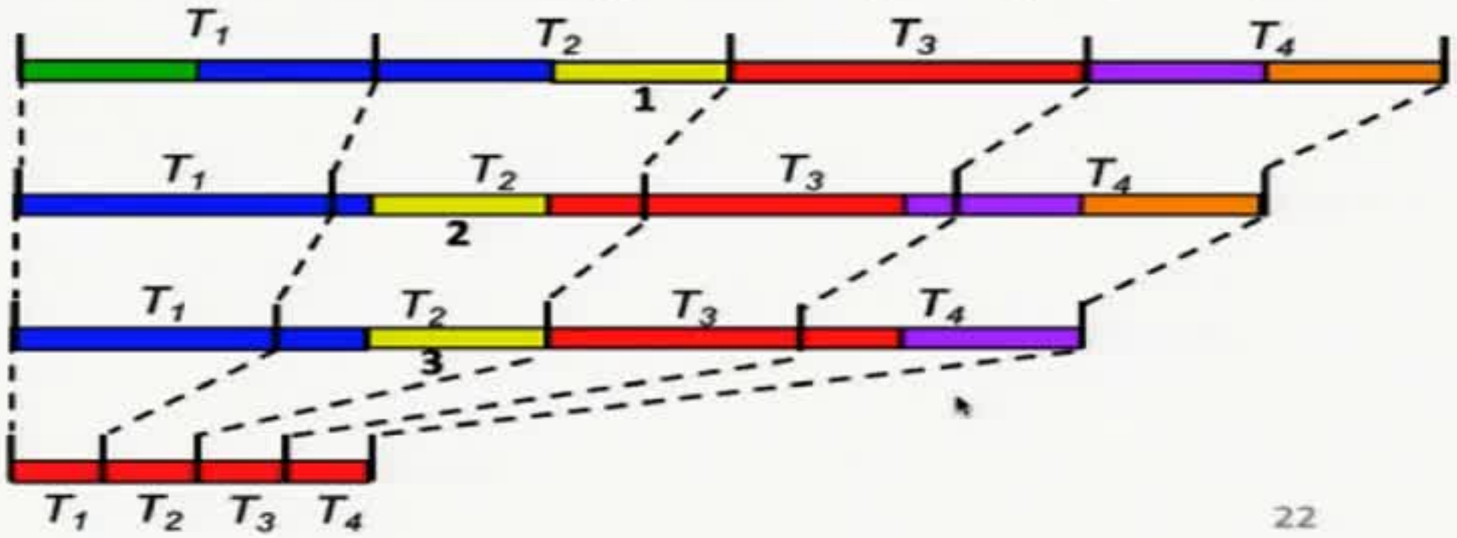
Target $\mathcal{H}$-matrix



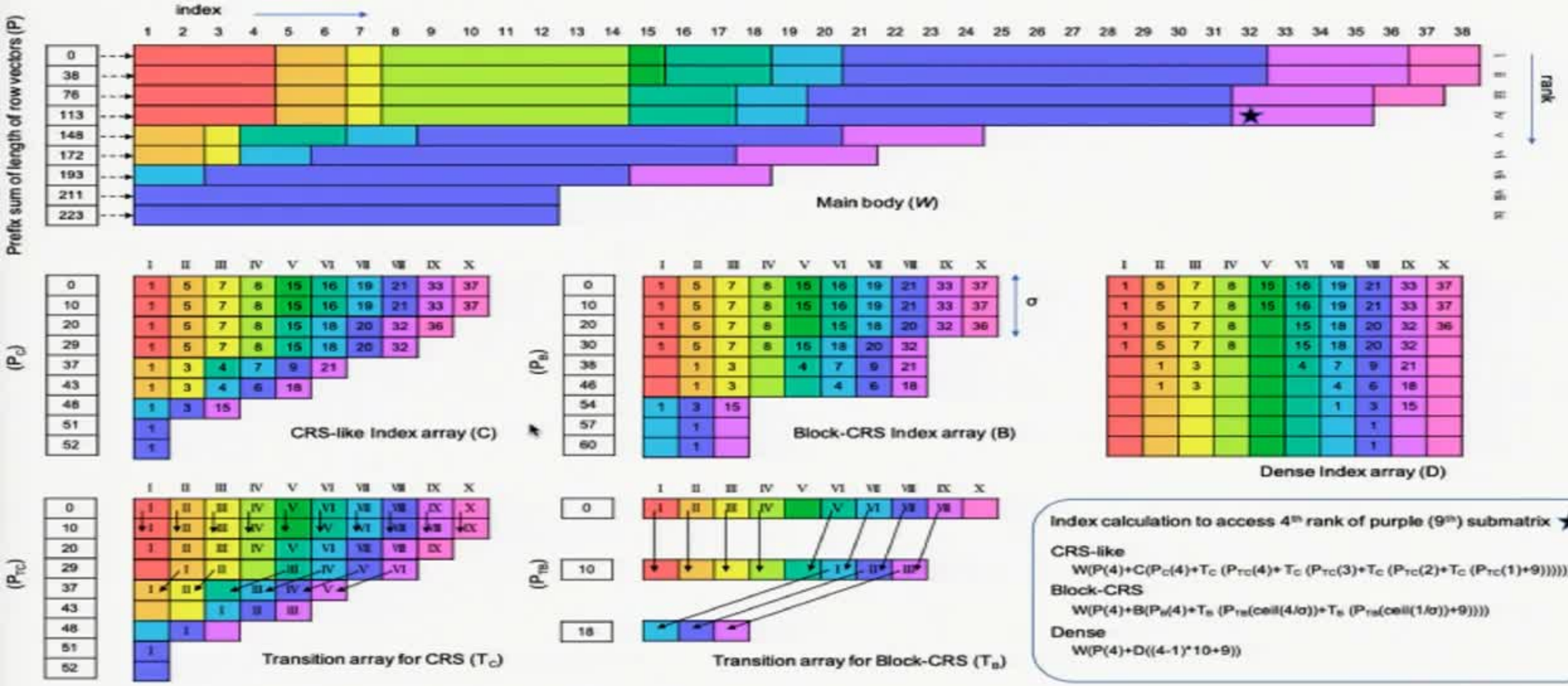
Locality aware layout (naïve)



Load balancing aware layout (proposal)



Index arrays for load-balancing-aware layout



Test cases

- Electric field analyses
- For calculate *i,j* element, each element requires...

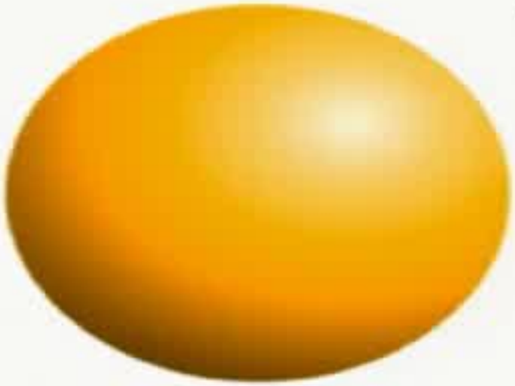
op	#
+	48
-	75
*	113
/	16
abs	2
sqrt	12
log	3
atan2	9

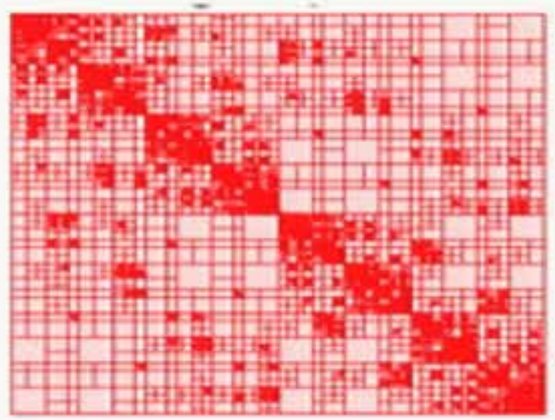
Each CUDA thread and SIMD element executes above

Input

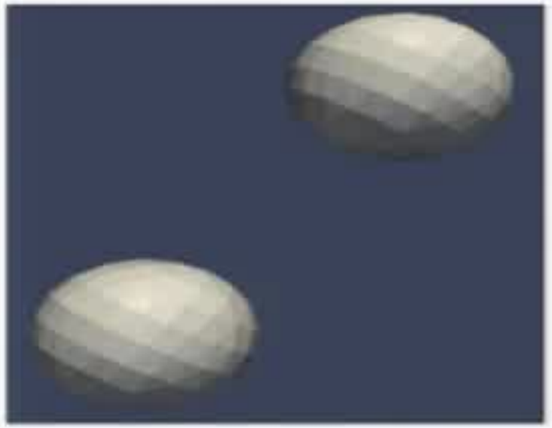
H-matrix structure

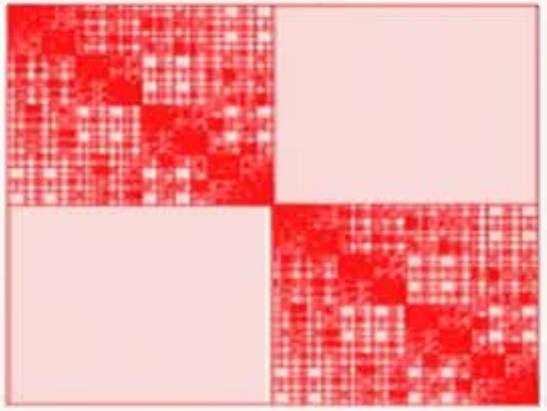
Single sphere



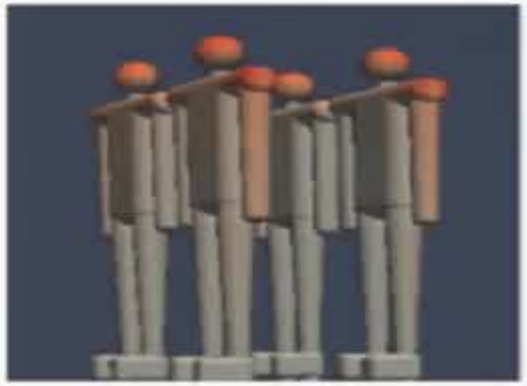


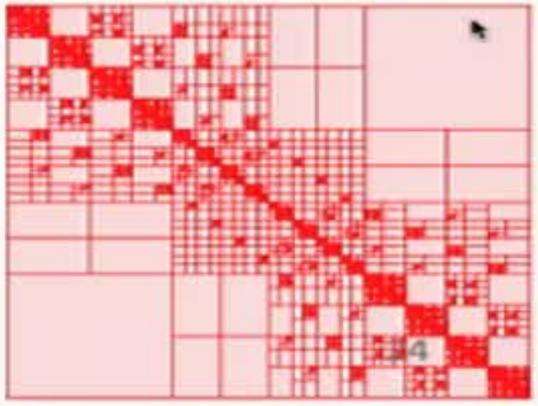
Double spheres





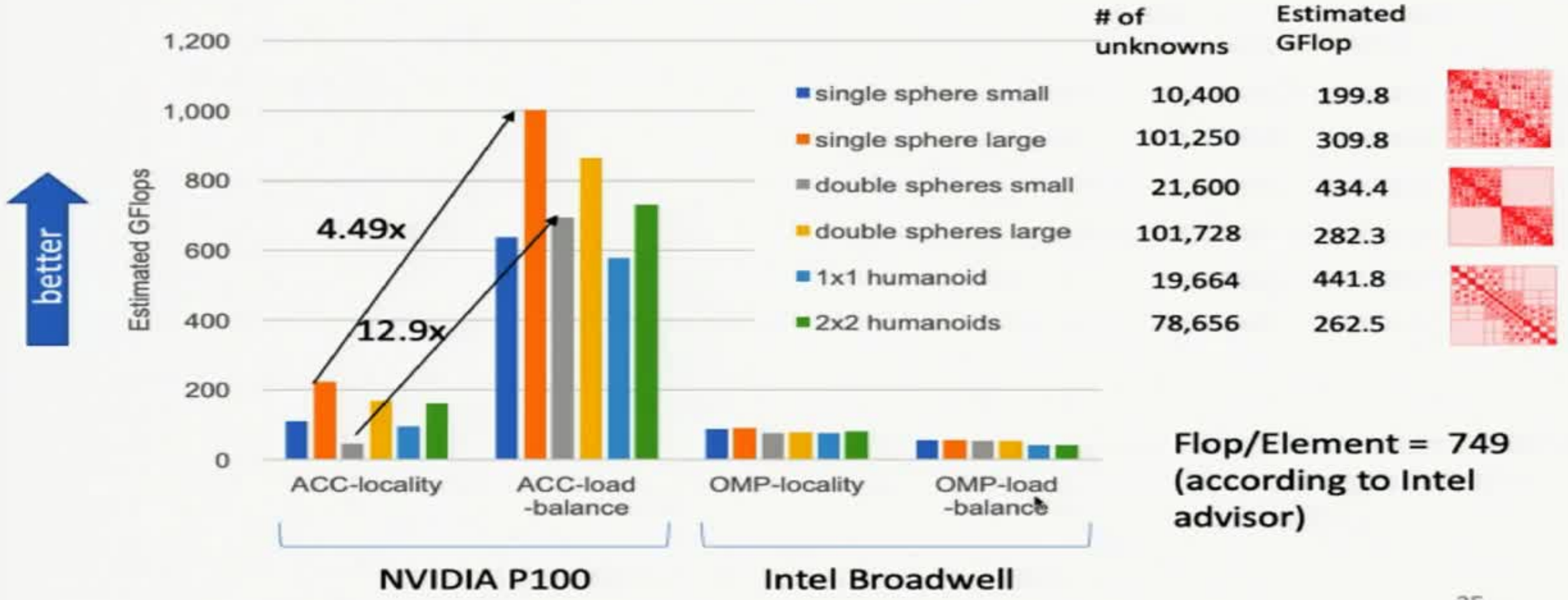
2x2 humanoids





$\mathcal{H}$-matrix Making with Electro-static Field Analysis

	P100	BDW
Peak performance [GFlops]	5,304	605

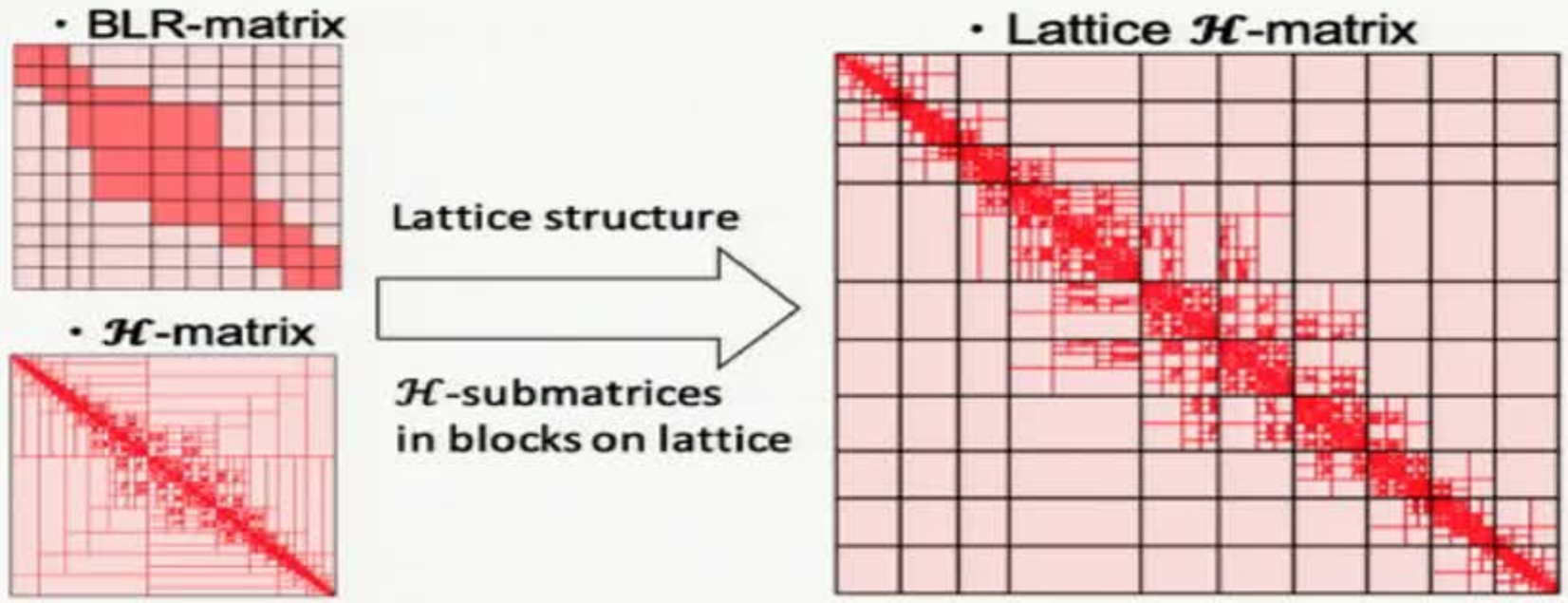


	# of unknowns	Estimated GFlop
single sphere small	10,400	199.8
single sphere large	101,250	309.8
double spheres small	21,600	434.4
double spheres large	101,728	282.3
1x1 humanoid	19,664	441.8
2x2 humanoids	78,656	262.5

Flop/Element = 749 (according to Intel advisor)

Current work

- Lattice $\mathcal{H}$-matrix [1]
  - ✓ Hybrid of BLR-matrix and $\mathcal{H}$-matrix
  - ✓ Load balancing method for distributed memory system
  - ✓ We are optimizing MPI communication of Lattice $\mathcal{H}$-matrix



[1] Akihiro Ida. Lattice H-matrices on distributed-memory systems (in press). In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2018.