User's and Reference Manual

Software Version: 1.0.0

Date: 2015/10/28

ACCMS, Kyoto University
JST CREST
e-mail: ida@media.kyoto-u.ac.jp

Contents

1 Introduction

This manual describes $\mathcal{H}$ACApK, a high performance software library for hierarchical matrices (H-matrices) [1,2,3,4] with adaptive cross approximation (ACA) [5,6]. The primary goal of the $\mathcal{H}$ACApK library is to provide implementations of H-matrices with ACA which work on massively parallel computers. The library includes parallel generators of H-matrices with ACA, parallel functions for performing arithmetic with H-matrices and parallel solvers for the solution of systems of linear equations whose coefficient matrix is expressed by an H-matrix [7].

1.1 Overview of Features

1.1.1 Target users

- **Researchers who deal with the N-body problem or simulation based on integral equation methods including BEM:** H-matrices with ACA adopted in $\mathcal{H}$ACApK can be applied to the coefficient matrices of systems of linear equations appeared in the evaluation of N-body problem or simulation based on the boundary element method (BEM).
- **Users of H-matrices libraries on serial computing:** If you already use any serial version of H-matrices library, you can easily enjoy the benefit of parallel computing by replacing your using library by $\mathcal{H}$ACApK.
- **From beginners to experts:** Users can utilize expertise by the use of $\mathcal{H}$ACApK. The beginners can use it with a minimal effort. The experts can take through parameters further control of generating H-matrices and solving systems of linear equations.

1.1.2 Short summary of H-matrices with ACA

In a narrow sense, H-matrices with ACA can be regarded as an approximation technique for matrices derived from BEM. Naïve application of BEM yields a dense matrix with full rank. The matrix requires memory footprints and computational costs of (N^2), where N denotes the degree of numbers of freedom. The complexity is reduced from (N^2) to $(N\log N)$ by using

H-matrices with ACA.

Generally, there are no explicit structures on the dense matrix derived from BEM. However, it is known that there are proper permutation and partition such that most of sub-matrices become numerically low-rank. Then sub-matrices detected as low-rank are approximated by using ACA. ACA is based on the idea that a low-rank matrix can be approximated by some pivot columns and pivot rows. By using H-matrices with ACA, the approximated matrix can be directly constructed WITHOUT making dense matrix.

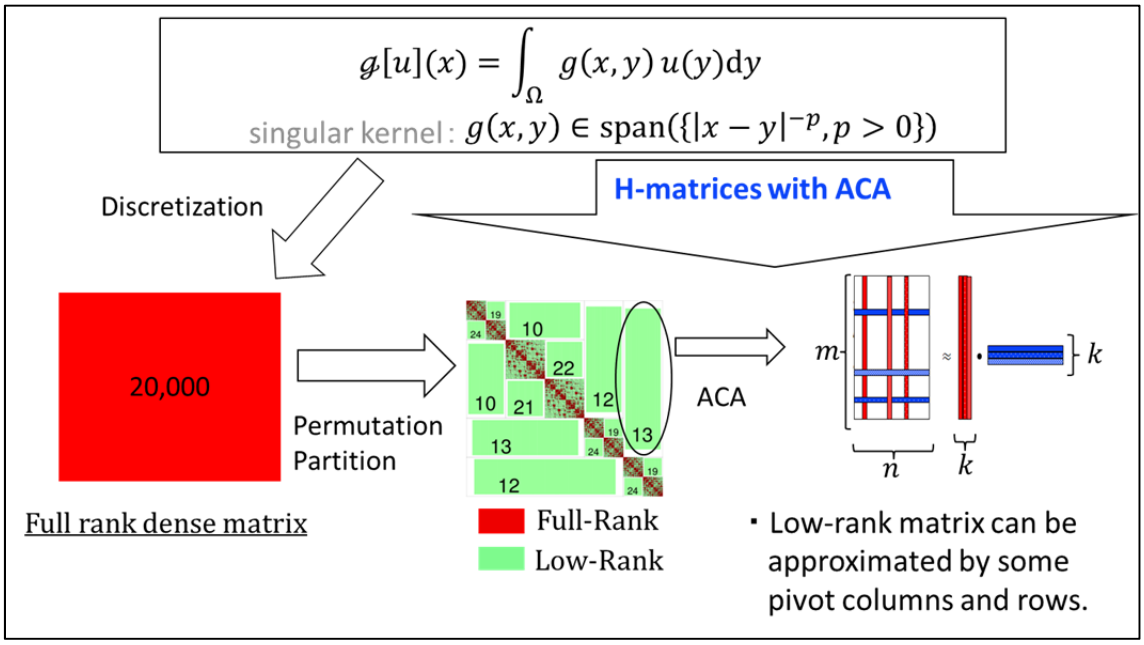


Figure 1.1 Approximated matrix by using H-matrices with ACA.

1.1.3 Advantages of H-matrices with ACA

As previously noted, Naïve application of BEM requires a large amount of memory usage because it yields a dense matrix. We can expect that H-matrices with ACA considerably reduce memory usage. In Figure 1.2, memory usage of H-matrices and original dense matrices are plotted as a function of the number of unknowns in the case of BEM for static electric field analysis. Logarithmic scales are used on both axes. Blue squares show the cases of the dense matrices, while red circles show the cases of HACApK. When we consider the use of 1 T-byte memory for storing matrices, we can handle only problem with half million unknowns in the case of the dense

matrix, while 30 million if we use H-matrix.

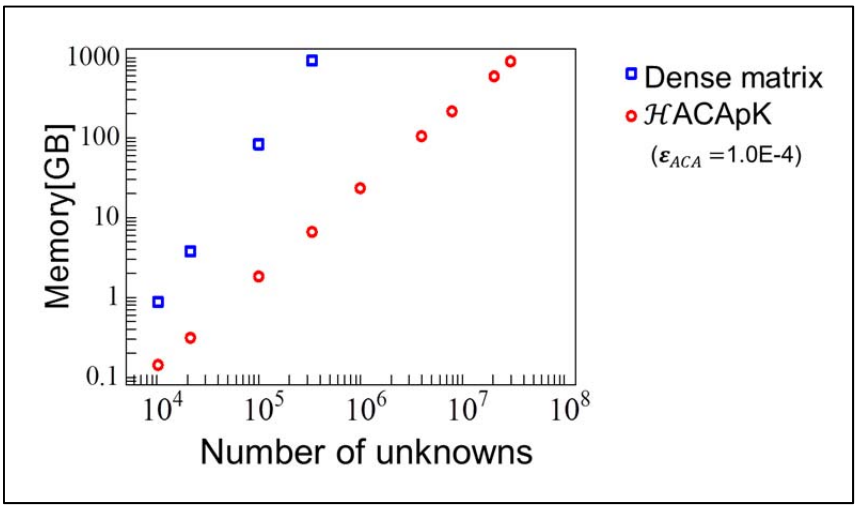


Figure 1.2 Comparison of memory usage

Another advantage of the use of H-matrices with ACA is that the approximation accuracy is controllable by a given tolerance $\varepsilon_{\mathrm{ACA}}$. When we set the tolerance as values in Table 1.1, measured approximation accuracy ε_{H} with Frobenius norm is little smaller than the given tolerance in all cases. It means the approximation accuracy is well controlled.

Table 1.1 Measured approximation accuracy for given tolerances

	Accuracy ε_H	
$\varepsilon_{\mathrm{ACA}}$	N=2,400	N=2,1600
1.0E - 3	3.57E−4	4.88E−4
1.0E - 4	3.33E - 5	5.03E−5
1.0E - 5	3.69E - 6	7.41E−6

$$\varepsilon_H = \frac{\|A - A_H\|_F}{\|A\|_F}$$

A : Original dense matrix

A_H : H-matrices approximation

$$\|A\|_F \equiv \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2}$$: Frobenius norm

1.2 Getting More Information

This manual consists of sections describing installation information and how to use the library. Moreover, the functions available in $\mathcal{H}$ACApK, which are the highest frequency of use in applications, are listed in the latter part of the manual. In addition to this manual, you will be able to get information from articles and presentations related to the $\mathcal{H}$ACApK library, which will be listed in the same website providing the library. Moreover, the manual of the software framework ppOpen-APPL/BEM for large-scale parallel Boundary Element Method analyses may be helpful. The manual is included in a set of files for 'BEM-BB-framework with $\mathcal{H}$ACApK'.

2 How to use the $\mathcal{H}$ACApK Library

2.1 Specifications

All programs of the $\mathcal{H}$ACApK library are written in Fortran 90 language and are provided in source code form. The MPI library is utilized for parallel computing on distributed memory computer systems, and OpenMP paradigm for parallel computing on shared memory. In order to use $\mathcal{H}$ACApK, users have to prepare the computer system installed **MPI** and the **Fortran 90 compiler** which can compile programs with OpenMP directives.

2.2 Installing $\mathcal{H}$ACApK

First, users should find a set of files for '$\mathcal{H}$ACApK library' and a set of files for 'BEM-BB-framework with $\mathcal{H}$ACApK'. The latter is a complete set to build an executable module for electrostatic analyses. The executable module can be built by simply typing '**make**' after user appropriately revises the Makefile according to the computer system. It is recommended that you build the executable module and run it before all other operations. You can confirm whether $\mathcal{H}$ACApK work on your system.

The $\mathcal{H}$ACApK library is not a complete set to build an executable module. For a complete set, users have to add the main program, which call subroutines of $\mathcal{H}$ACApK, and the user function which is called from $\mathcal{H}$ACApK (see

subsection 2.3). For compiling the complete set, user have to take care of makedependency. The Makefile included 'Bem-bb-framework with $\mathcal{H}$ACApK' may prove helpful.

2.3 Providing user function

In order to use the $\mathcal{H}$ACApK, users should provide the functions that we call 'user function'. $\mathcal{H}$ACApK requires the function to calculate an element of the dense matrix that is approximated by H-matrices with ACA. In the case of the boundary element method, the element will be given as the result of the integral operations conducted between a pairs of the boundary elements. The file 'm_HACApK_calc_entry_ij.f90', which is included the set of files 'BEM-BB-framework with $\mathcal{H}$ACApK', may prove helpful as an example of the function. The user function in the files is for the surface charge method for a static electric field problem. For the detail of the problem supported by the user function, refer to the manual of 'BEM-BB-framework' and section 3.

The user function must be named '`HACApK_entry_ij`', and return a double precision value as a required element. The user function has a fixed form, and should be written in the following form:

```
real*8 function HACApK_entry_ij(i,j,st_bemv)
 integer :: i,j
 type(st_HACApK_calc_entry) :: st_bemv
    ⋮
 return HACApK_entry_ij
end function
```

where the words written in the bold font denote that these words are parts of the fixed form. The integer argument '`i`' designates the ordinal number of rows of the dense matrix, and the integer argument '`j`' of columns, respectively. The third argument '`st_zbemv`' of a structure type is used to store data needed for calculating the value of the element. The type name of the structure should be '`st_HACApK_calc_entry`'. The way to describe the structure also has a fixed form, and should be written in the following form:

```
type(st_HACApK_calc_entry)
 integer :: nd,lp61
 real*8,pointer :: ao(:)
 ------------------------------
 integer ::  int_ex
 real*8  ::  dbl_ex
 integer,pointer :: int_ex1(:),int_ex2(:,:)
 real*8,pointer  :: dbl_ex1(:),dbl_ex2(:,:)
end function
```

where the variables written above the line, which are the integer variables 'nd' 'lp61' and the real*8 pointer 'ao', are reserved words of $\mathcal{H}$ACApK. Users should declare these words, and should not use them for their code. Users can declare any type of variables (scalar, array, pointer, structure) as much as needed for calculations in the user function. For example, the variables may have information of coordinates of points associated with the ordinal number of the matrix.

2.4 Writing your code

The pseudo code 2.4.1 illustrates the basic usage of the $\mathcal{H}$ACApK. It consists of following 4 steps including function calls to $\mathcal{H}$ACApK:

1. **Initialize $\mathcal{H}$ACApK**. The function 'HACApK_init' allocates members of some structure type variables used in $\mathcal{H}$ACApK, and sets default parameters for H-matrices with ACA.
2. **Generate an H-matrix** by the function 'HACApK_generate'.
3. **Utilize the H-matrix**. Currently, we provide a function to perform H-matrix-vector multiplication and a function to solve the system of linear equations by using Bi-CGSTAB method. In the pseudo code, the liner solver 'HACApK_solve' is called.
4. **Finalize $\mathcal{H}$ACApK**.

Although users should declare structure type variables used in $\mathcal{H}$ACApK, not need to explicitly allocate or set these values by themselves. All these variables are appropriately set by calling functions of $\mathcal{H}$ACApK in sequence. Arrays and integers appeared in the pseudo code are probably understood by

referring to the reference guide in section 4, except the real*8 array '`coord`'. We describe the array in next subsection.

In the case that your program utilizes the MPI library, the MPI function '`MPI_Init`' should be called in your code before step 1, and '`MPI_Finalize`' after step 4. The pseudo code 2.4.2 illustrates the case. Note that arguments of the function '`MPI_Init`' includes '`icomm`' which is not needed in 2.4.1.

```
 program Example_Using_HACApK
 use m_HACApK_solve
 use m_HACApK_base
 implicit real*8(a-h,o-z)
 type(st_HACApK_lcontrol) :: st_ctl
 type(st_HACApK_leafmtxp) :: st_leafmtxp
 type(st_HACApK_calc_entry) :: st_bemv
 real*8,dimension(:,:),allocatable :: coord
 real*8,dimension(:),allocatable :: rhs,sol
    ⋮
 <<< User code part 1 >>>
    ⋮
 lrtrn=HACApK_init(nd,st_ctl,st_bemv)
 allocate(coord(nd,3),rhs(nd),sol(nd))
    ⋮
 <<< User code part 2 >>>
    ⋮
 ztol=1.0e-5
 lrtrn=HACApK_generate(st_leafmtxp,st_bemv,st_ctl,coord,ztol)
 lrtrn=HACApK_solve(st_leafmtxp,st_bemv,st_ctl,rhs,sol,ztol)
 lrtrn=HACApK_free_leafmtxp(st_leafmtxp)
 lrtrn=HACApK_finalize(st_ctl)
    ⋮
 <<< User code part 3 >>>
    ⋮
end program
```

Pseudo code 2.4.1: example of the basic usage of the $\mathcal{H}$ACApK.

```
 program Example_Using_HACApK
 use m_HACApK_solve
 use m_HACApK_base
 implicit real*8(a-h,o-z)
 include 'mpif.h'
 type(st_HACApK_lcontrol) :: st_ctl
 type(st_HACApK_leafmtxp) :: st_leafmtxp
 type(st_HACApK_calc_entry) :: st_bemv
 real*8,dimension(:,:),allocatable :: coord
 real*8,dimension(:),allocatable :: rhs,sol
 icomm=MPI_COMM_WORLD; ierr=0
 call MPI_Init (ierr)
    ⋮
 <<< User code part 1 >>>
    ⋮
 lrtrn=HACApK_init(nd,st_ctl,st_bemv,icomm)
 allocate(coord(nd,3),rhs(nd),sol(nd))
    ⋮
 <<< User code part 2 >>>
    ⋮
 ztol=1.0e-5
 lrtrn=HACApK_generate(st_leafmtxp,st_bemv,st_ctl,coord,ztol)
 lrtrn=HACApK_solve(st_leafmtxp,st_bemv,st_ctl,rhs,sol,ztol)
 lrtrn=HACApK_free_leafmtxp(st_leafmtxp)
 lrtrn=HACApK_finalize(st_ctl)
    ⋮
 <<< User code part 3 >>>
    ⋮
 call MPI_Finalize (ierr)
end program
```

Pseudo code 2.4.2: example of the usage of the $\mathcal{H}$ACApK in the case that the user' program utilizes the MPI library.

2.5 Coordinates information

As mentioned before, H-matrices can be regarded as a method for finding out proper permutation and partition on a dense matrix such that most of sub-matrices become numerically low-rank. Although several variants for the method are proposed [3,4], the $\mathcal{H}$ACApK employs the method which utilizes spatial coordinates associated with row (or column) indices of the original matrix. See subsections 2.5.1 and 2.5.2 about how to choose the point whose coordinates are used in $\mathcal{H}$ACApK.

You should store the coordinates with a two-dimensional array, which corresponds to the array '`coord`' in pseudo code 2.4.1, and pass the array to the function '`HACApK_generate`'. In pseudo code, the size of dimensions of the array are determined as '`coord(nd,3)`'. The size of first dimension, '`nd`', should coincide with the number of row indices of the matrix, and the size of second dimension should be '3' because the three-dimensional space of Euclidean geometry is expected in $\mathcal{H}$ACApK. The variable `coord(i,1)` should preserve the x-coordinate of the point associated with the i-th row index of the matrix, and the variable `coord(i,2)`the y-coordinate, and the variable `coord(i,3)`the z-coordinate. If you deal with the two-dimensional analyses, you should set all values of `coord(i,3)` to zero.

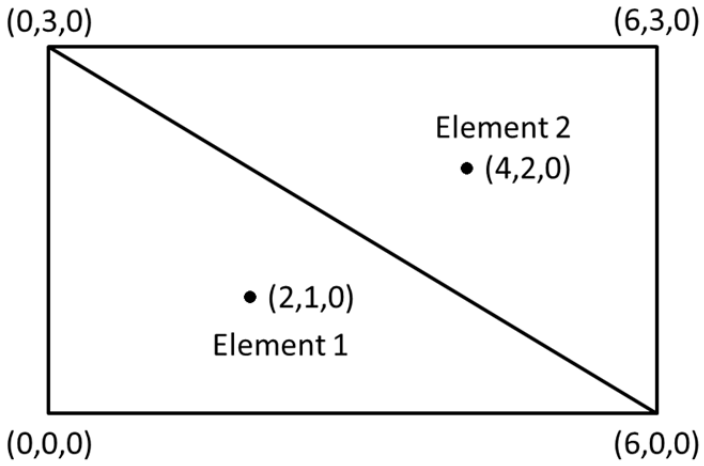
2.5.1 For N-body problem

In the case of N-body problem, each row index number of the matrix will be associated on one-to-one correspondence with each body. One possibility is that the coordinates required by $\mathcal{H}$ACApK for permutation and partition is given by the center of gravity of each body.

2.5.2 For boundary element method

We here consider the case that $\mathcal{H}$ACApK is applied to the BEM analyses. If step functions are used as the base functions of BEM, each row index number of the matrix will be associated on one-to-one correspondence with each element. We can choose the center of balance on each element as the point whose coordinates are used in $\mathcal{H}$ACApK(see Fig.2.1). If higher order functions are employed as the base functions, there are some functions associated with row indices on each element. Then the row index number

and the element are not in one-to-one correspondence because a row index number is associated with a base function. Even in this case, one possibility is that the center of balance on each element gives the coordinates for all base functions on the element. Another possibility is that the coordinates are shifted from the center of balance with consideration to symmetry aberration of the base function on an element.




```
nd=2
coord(1,1)=2.0; coord(1,2)=1.0; coord(1,3)=0.0
coord(2,1)=4.0; coord(2,2)=2.0; coord(2,3)=0.0
```

Figure 2.1 Example of how to give coordinates information in the case of BEM by using step base functions as the base function.

3 Application examples and performance evaluations

3.1 Linear System Derived from Integral Equations with Singular Kernels

We now describe the typical formulation of the system of linear equations such that H-matrices can be successfully applied. Let H be a Hilbert space of functions on a $(d-1)$-dimensional domain $\Omega \subset \mathbb{R}^d$, and H' be the dual space of H. For $u \in H, f \in H'$ and a kernel function of a convolution operator $g: \mathbb{R}^d \times \Omega \to \mathbb{R}$, we consider an integral equation

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

To numerically calculate Equation 2, we divide the domain, Ω, into elements $\Omega^h = \{\omega_j : j \in J\}$, where J is an index set. When we use the 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 $(\varphi_i)_{i\in ב}$ of H^h for an index set $ב := \{1, \cdots, N\}$, the approximant $u^h \in H^h$ to u can be expressed using a coefficient vector $\phi = (\phi_i)_{i\in ב}$ that satisfies $u^h = \sum_{i\in ב} \phi_i \varphi_i$. Note that the supports of the basis $\Omega^h_{\varphi_i} := \operatorname{supp} \varphi_i$ are assembled from the sets ω_j. Equation 2 is reduced to the following system of linear equations.

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

In the case of the Ritz-Galerkin method, the entries of A and B are given by

$$A_{ij} = \int_{\Omega} \varphi_i(x) \int_{\Omega} g(x,y)\,\varphi_j(y)\mathrm{d}y\mathrm{d}x \qquad \text{for all } i,j \in ב \tag{3}$$

$$B_i = \int_{\Omega} \varphi_i(x)\, f \mathrm{d}x \qquad \text{for all } i \in ב \tag{4}$$

Suppose that the kernel functions are written in the form

$$g(x,y) \in \operatorname{span}(\{|x-y|^{-p}, p > 0\}). \tag{6}$$

Moreover, permute the index set $ב := \{1, \cdots, N\}$. Then there exist H-matrices that efficiently approximate the coefficient matrix $A \in \mathbb{R}^{ב\times ב}$. Such kernel functions appear in a number of scientific applications, for example, electric field analyses, mechanical analyses, and earthquake cycle simulation. It is well known that the N-body problem, whose formulation is different from the one above, results in a similar matrix.

For more practical example, we here consider an electrostatic field problem. We suppose that a perfect conductor, which has the shape of a sphere, is set in a space such that the distance between the center of the sphere and the surface of the ground with zero electric potential is 0.5m. The radius of the sphere conductor is 0.25 m, and the conductor is charged such that its electric potential becomes 1V. We employ the surface charge method to calculate the electrical charge on the surface of the conductor. The surface charge is supposed to be distributed as shown in Figure 3.1. When applying BEM to the above electrostatic field analyses, we can use the formulation

described in Section 3.1, with the kernel function given by

$$g(x,y) = \frac{1}{4\pi\epsilon}|x-y|^{-1} \tag{7}$$

Here, ϵ denotes the electric permittivity. When we divide the surface of the conductor into triangular elements (see Figure 3.1), and use step functions as the base function of BEM, we can apply H-matrices to the coefficient matrices of the systems of linear equations derived from the above formulation.

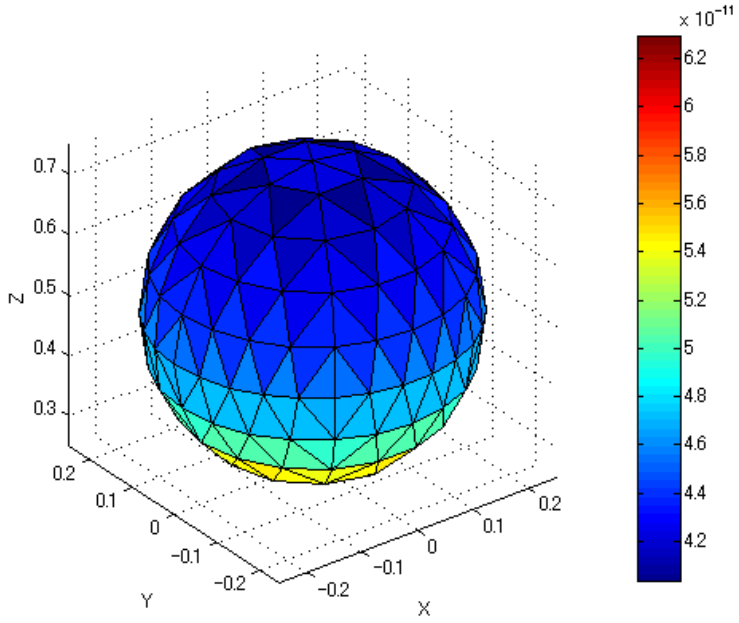


Figure 3-1. The calculated surface charge density

3.2 Parallel scalability of flat-MPI version

We applied HACApK to the problem shown in the latter part of section 3.1. Strong scaling experiments were carried out for meshes with the number of element N=1000, 10400 and 101250. We used a Fujitsu FX10 at the University of Tokyo for the test analyses. The error tolerance for ACA was set as ε=1.0e-4. Figure 3-2 shows the speed-up versus the execution time of the serial computing in the H-matrix construction step using HACApK, while Figure 3-3 shows the result for one H-matrix-vector multiplication(HMVM). They have been plotted as a function of the number of processes. The larger

the data size becomes, the better parallel scalability HACApK attains in both the H-matrix construction and HMVM steps.

In the H-matrix construction, a more than 50-fold speedup is obtained by using 64 cores for the largest model (N=101,250). This is because MPI communication is not necessary in the H-matrix construction algorithm. Moreover, because the calculation of the kernel function for the construction step is computationally intensive, the parallel speedup ratio is not overly affected by the memory bandwidth. We consider the difference between the ideal and obtained speedups to be due to the load-imbalance among the MPI processes.

Conversely, we observed limits in the speed-up ratios for HMVM. Although the attained maximum speedup ratio becomes higher as the data size becomes larger, the speed-up ratio seems to have reached its limit when 64 MPI-processors are used, even for the largest data. For HMVM, we pay the cost of parallelization through the MPI communication and waiting costs. We believe that the ceiling is caused by the MPI communication costs because the load-imbalance does not significantly deteriorate the parallel speed-up ratio when constructing H-matrices.

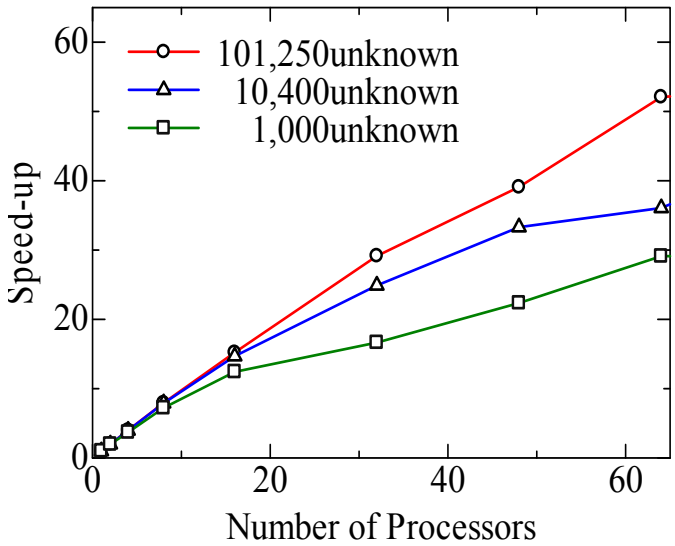


Figure 3-2. Parallel scalability when constructing H-matrices

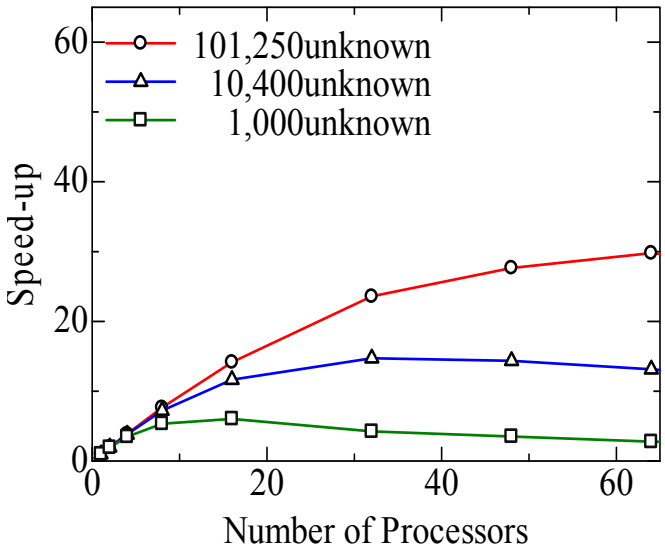


Figure 3-3. Parallel scalability when performing an HMVM

3.3 Parallel scalability of hybrid MPI+OpenMP version

The functions for the H-matrix generation and HMVM by using hybrid MPI+OpenMP programming approach will be available after next update of HACApK library. The parallel scalability when performing HMVM will be considerably improved from flat-MPI version.

4 Reference Guide

4.1 Structures

Here are defined some structure types used in function and subroutines of the $\mathcal{H}$ACApK library.

4.1.1 st_HACApK_calc_entry

This structure type must be defined by the user (see 2.3).

4.1.2 st_HACApK_leafmtxp

Define the `st_HACApK_leafmtxp` representation which expresses leaves of an H-matrix. The member `nlf` represents the number of sub-matrices of the H-matrix, and `nlfkt` the number of sub-matrices expressed by low-rank

decomposed matrices. The entries of sub-matrices will be stored in the array of structure `st_lf(:)`. See 4.1.3 about the element of the array of structure.

Declaration

```
type :: st_HACApK_leafmtxp
   integer*4 nd,nlf,nlfkt,ktmax
   type(leafmtx),pointer :: st_lf(:)
end type st_HACApK_leafmtxp
```

4.1.3 st_HACApK_leafmtx

Define data structure whose variable expresses a sub-matrix of the H-matrix.

Declaration

```
type :: st_HACApK_leafmtx
   integer*4 ltmtx
   integer*4 kt
   integer*4 nstrtl,ndl
   integer*4 nstrtt,ndt
   real*8,pointer :: a1(:,:),a2(:,:)
 end type st_HACApK_leafmtx
```

4.1.4 st_HACApK_lcontrol

Define data structure whose members are used for controlling the generation of H-matrices and the way to perform the parallel computing.

Declaration

```
type :: st_HACApK_lcontrol
  integer*4,pointer :: lod(:),lsp(:),lnp(:),lthr(:),lpmd(:)
  real*8,   pointer :: param(:)
  integer :: lf_umpi
end type st_HACApK_lcontrol
```

4.2 Functions and Subroutines

The $\mathcal{H}$ACApK library has a large number of functions and subroutines. However, here are described only some functions and subroutines that are expected with high frequency of use in application.

4.2.1 HACApK_init

Set up the environment to work other functions of the $\mathcal{H}$ACApK library. This function should be called before any other function of the $\mathcal{H}$ACApK library is called. The declaration of using the module `m_HACApK_base` is needed for using this function.

Usage

```
use m_HACApK_base
type(st_HACApK_lcontrol) :: st_ctl
type(st_HACApK_calc_entry) :: st_bemv
integer :: lrtrn, icomm, nd

lrtrn = HACApK_init(nd,st_ctl,st_bemv[,icomm])
```

Results

`lrtrn` : is zero if successful; otherwise a positive integer which means an error code.

Input Arguments

`nd` : specifies the number of rows of the matrix approximated by H-matrices.

`st_bemv` : contains information for calculating entry values of the matrix approximated by H-matrices. The information must be set by the user.

`icomm` : is the optional argument. If the user's program utilizes the MPI library, the argument '`icomm`' specifies a MPI communicator. If it is not the case, the argument '`icomm`' should be omitted.

Output Arguments

`st_ctl` : contains parameters for controlling the $\mathcal{H}$ACApK library.

`st_bemv` : Some members of the structure variable are set.

4.2.2 HACAkK_generate

Generate H-matrices in the `st_HACApK_leafmtxp` representation. The

function HACApK_init should be called before this function is called. The declaration of using the module 'm_HACApK' is needed for using this function.

Usage

```
use m_HACApK
type(st_HACApK_leafmtxp) :: st_leafmtxp
type(st_HACApK_lcontrol) :: st_ctl
type(st_HACApK_calc_entry) :: st_bemv
real*8 :: coord(nd,3), ztol

lrtrn=HACApK_generate_leafmtx(st_leafmtxp,st_bemv,st_ctl,
                              coord,ztol)
```

Results

lrtrn : is zero if successful; otherwise a positive integer which means an error code.

Input Arguments

coord : must be an array, dimension(nd,3), where nd is the number of rows of the matrix approximated by H-matrices.

st_bemv : contains information for calculating entry values of the matrix approximated by H-matrices. The information must be set by the user.

st_ctl : will be adequately set by calling the function HACApK_init.

ztol : defines the tolerance that controls the approximation accuracy of the H-matrices in the meanings of the relative error with Frobenius-norm.

Output Arguments

st_leafmtxp: contains the matrix in the st_HACApK_leafmtxp representation which is approximated by H-matrices.

4.2.3 HACApK_solve

Solve the system of linear equations whose coefficient matrix is stored in the st_HACApK_leafmtxp representation. The function HACApK_init should be called before this function is called. The declarations of using the modules, m_HACApK and m_krylov_hmtx, are needed for using this function.

Usage

```
use m_HACApK
use m_krylov_hmtx
type(st_HACApK_leafmtxp) :: st_leafmtxp
type(st_HACApK_lcontrol) :: st_ctl
type(st_HACApK_calc_entry) :: st_bemv
real*8 :: rhs(nd),sol(nd),ztol

lrtrn = HACApK_solve_leafmtx(st_leafmtxp,st_bemv,st_ctl,
                             rhs,sol,ztol)
```

Results

lrtrn : is zero if successful; otherwise a positive integer which means an error code.

Input Arguments

st_leafmtxp: should contain the matrix in the st_HACApK_leafmtxp representation.

st_bemv : should contain information for calculating entry values of the matrix approximated by H-matrices. The information must be set by the user.

st_ctl : will be adequately set by calling the function HACApK_init.

rhs : must be an array, dimension(nd), where nd is the number of linear equations. The array should hold the values of the right hand side of the system of linear equations.

sol : must be an array, dimension(nd). The values in this array will be used as the approximate initial solution of an iterative solver for the linear systems.

ztol : defines the tolerance that controls the approximation accuracy of the solution in the meanings of the relative residual with the 2-norm.

Output Arguments

sol : will hold the solution of the system of linear equations when returned from this function.

4.2.4 HACAkK_adot_pmt_lfmtx_p

Perform the multiplication of an H-matrix and a vector, where the H-matrix must be stored in the st_HACApK_leafmtxp representation. The function HACApK_init should be called before this subroutine is called. The declarations of using the modules, m_HACApK and m_krylov_hmtx, are needed for using this function.

Usage

```
use m_HACApK
use m_krylov_hmtx
type(st_HACApK_leafmtxp) :: st_leafmtxp
type(st_HACApK_lcontrol) :: st_ctl
type(st_HACApK_calc_entry) :: st_bemv
real*8 :: ax(nd),x(nd),lrtrn

lrtrn = HACAkK_adot_pmt_lfmtx_p (st_leafmtxp,st_bemv,st_ctl,
                                 ax,x)
```

Results

lrtrn : is zero if successful; otherwise a positive integer which means an error code.

Input Arguments

st_leafmtxp: should contain the matrix in the st_HACApK_leafmtxp representation.

st_bemv : should contain information for calculating entry values of the matrix approximated by H-matrices. The information must be

set by the user.

st_ctl : will be adequately set by calling the function HACApK_init.

x : must be an array, dimension(nd), where nd is the number of rows of the matrix approximated by H-matrices. The array should hold the multiplicand vector.

Output Arguments

ax : will hold the result of the H-matrix-vector multiplication when returned from this subroutine.

lrtrn : is zero if successful; otherwise a positive integer which means an error code.

4.2.5 HACApK_adot_pmt_lfmtx_hyp

Perform the multiplication of an H-matrix and a vector by using hybrid MPI+OpenMP programing model, where the H-matrix must be stored in the st_HACApK_leafmtxp representation. The function HACApK_init should be called before this subroutine is called. The declarations of using the modules, m_HACApK and m_krylov_hmtx, are needed for using this function.

Usage

```
use m_HACApK
use m_krylov_hmtx
type(st_HACApK_leafmtxp) :: st_leafmtxp
type(st_HACApK_lcontrol) :: st_ctl
type(st_HACApK_calc_entry) :: st_bemv
real*8 :: ax(nd),x(nd),lrtrn

lrtrn =HACApK_adot_pmt_lfmtx_hyp(st_leafmtxp,st_bemv,
                                 st_ctl,ax,x)
```

Results

lrtrn : is zero if successful; otherwise a positive integer which means an error code.

Input Arguments

st_leafmtxp: should contain the matrix in the st_HACApK_leafmtxp representation.

st_bemv : should contain information for calculating entry values of the matrix approximated by H-matrices. The information must be set by the user.

st_ctl : will be adequately set by calling the function HACApK_init.

x : must be an array, dimension(nd), where nd is the number of rows of the matrix approximated by H-matrices. The array should hold the multiplicand vector.

Output Arguments

ax : will hold the result of the H-matrix-vector multiplication when returned from this subroutine.

lrtrn : is zero if successful; otherwise a positive integer which means an error code.

4.2.6 HACAkK_free_leafmtxp

Deallocate arrays and pointers allocated as the members of the structure type st_HACApK_leafmtxp. The declaration of using the module m_HACApK is needed for using this function.

Usage

```
use m_HACApK
type(st_HACApK_leafmtxp) :: st_leafmtxp

lrtrn =HACApK_leafmtxp(st_leafmtxp)
```

Results

lrtrn : is zero if successful; otherwise a positive integer which means an error code.

4.2.7 HACAkK_finalize

Terminates HACApK execution environment. This function deallocates pointers allocated as the member of the structure type `st_HACApK_lcontrol`. The declaration of using the module `m_HACApK` is needed for using this function.

Usage

```
use m_HACApK
type(st_HACApK_lcontrol) :: st_ctl

lrtrn=HACApK_lcontrol(st_ctl)
```

Results

`lrtrn` : is zero if successful; otherwise a positive integer which means an error code.

5 List of references

1. Hackbusch W.: A Sparse Matrix Arithmetic Based on H-matrices. I. Introduction to H-matrices, Computing, Vol. 62(2), pp. 89-108 (1999).
2. Hackbusch W. and Khoromskij B.N.: A Sparse H-matrix Arithmetic. II. Application to Multi-dimensional Problems, Computing, Vol. 64(1), pp. 21-47 (2000).
3. Börm S., Grasedyck L. and Hackbusch W.: Hierarchical Matrices, Lecture Note, Max-Planck-Institut fur Mathematik, (2006).
4. Bebendorf M.: Hierarchical Matrices, Springer(2008).
5. Kurtz S., Rain O. and Rjasanow S.: The Adaptive Cross-Approximation Technique for the 3-D Boundary-Element Method, IEEE Trans. Magn., Vol.38(2), pp.421-424 (2002).
6. Bebendorf M. and Rjasanow S.: Adaptive Low-Rank Approximation of Collocation Matrices, Computing, Vol. 70(1), pp.1-24 (2003).
7. Ida A., Iwashita T., Mifune T. and Takahashi Y.: Parallel Hierarchical Matrices with Adaptive Cross Approximation on Symmetric Multiprocessing Clusters, Journal of Information Processing, Vol. 55(4), (2014)