Parallel H-Matrices with Adaptive Cross Approximation for Large-Scale Simulation

A. Ida and T. Iwashita
(Academic Center for Computing and Media studies, Kyoto University)

February 20, 2014, Portland, Oregon, USA
Outline

1 Background & Objectives
2 New H-matrices code: $\mathcal{H}$ACApK
3 Parallelization of H-matrices in $\mathcal{H}$ACApK
4 Summary and Future work
ppOpen-APPL/BEM

Software framework for large-scale parallel BEM analyses

- The framework supports necessary communications on SMP cluster systems.
- User can easily modify the program and reflect their own ideas to it.
- Download the framework: [http://ppopenhpc.cc.u-tokyo.ac.jp/wordpress/](http://ppopenhpc.cc.u-tokyo.ac.jp/wordpress/)

Target applications

- Electromagnetic fields
- Earthquake cycle
- Quantum mechanics

Images:
- Transcribed from Ohtani et al. (2011)
- Transcribed from NIST Image Gallery
To overcome the disadvantage of BEM

Boundary Element Method (Integral equation method)
- Naïve application of BEM yields dense matrices.
  - memory footprints and computational costs of $\mathcal{O}(N^2)$
- Remedies are necessary for large-scale analyses.
  1. Approximation techniques for matrices: $\mathcal{O}(N^2) \Rightarrow \mathcal{O}(N\log^p N)$
  2. Parallel computing

Remedies in ppOpen-APPL/BEM
- The $H\text{ACApK}$ library will be available in next version.
  - It adopts $H$-matrices with ACA as an approximation technique.
- Hybrid MPI+OpenMP programming model is used to exploit SMP cluster systems.
Development of HACApK library

Objectives

- Develop software for H-matrices with ACA on SMP cluster systems
- Make large-scale BEM analyses possible
  - Hybrid MPI+OpenMP programming model
  - Open source library

- Available H-matrices software:
  - 'Hlib' (by Max-Plank-Institute), 'AHMED' (by Prof. M. Bebendorf)
  - Serial computation or flat-MPI approach
  - Commercial use is not allowed
1 Background & Objectives
2 New H-matrices code $\mathcal{H}$ACApK
3 parallelization of H-matrices in $\mathcal{H}$ACApK
4 Future work
H-matrices with ACA

An approximation technique for matrices derived from BEM.

\[ g[u](x) = \int_\Omega g(x, y) u(y) dy \]

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

Naïve application of BEM

Full rank dense matrix

Permutation Partition

ACA

- Low-rank matrix can be approximated by some pivot columns and rows.
Adaptive Cross Approximation (ACA)

- ACA reduces computational costs and memory usage.

- Number of entries:
  \[ \sigma(mn) \Rightarrow \sigma(k(m + n)), \]
  \[ k \ll m, n \]

- Matrix-vector product:

- The number \( k \) is automatically determined by ACA such that the approximation error is less than a given tolerance \( \varepsilon_{ACA} \).

- ACA is an algebraic algorithm.
  - The kernel function is not needed to be known concretely.
  - This nature of ACA is suitable for frameworks and libraries.
Test model: static electric field analysis

Surface charge is calculated in half-infinite domain.

- 1V is given to the conductor.
- The radius of the sphere is 0.25m.

Potential operator is given by

$$V[u](x) := \int_{\Gamma} \frac{1}{4\pi \|x-y\|} u(y) dy, \quad x \in \Gamma$$

Numerical result:
Surface charge density
Accuracy of the H-matrices with ACA

Approximation accuracy is controllable by tolerance $\varepsilon_{ACA}$.

<table>
<thead>
<tr>
<th>$\varepsilon_{ACA}$</th>
<th>$N=2,400$</th>
<th>$N=2,1600$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0E - 3</td>
<td>3.57E - 4</td>
<td>4.88E - 4</td>
</tr>
<tr>
<td>1.0E - 4</td>
<td>3.33E - 5</td>
<td>5.03E - 5</td>
</tr>
<tr>
<td>1.0E - 5</td>
<td>3.69E - 6</td>
<td>7.41E - 6</td>
</tr>
</tbody>
</table>

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

$A$ : Original dense matrix

$A_H$ : H-matrices approximation

$\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{i,j}|^2}$ : Frobenius norm
H-matrices with ACA reduce memory usage as expected.

Memory usage (Log-Log scale)

- Dense matrix
- $\mathcal{H}$ACA$pK$ ($\varepsilon_{ACA} = 1.0E-4$)
Two major functions of $\mathcal{H}$ACA$pK$

Hereafter, we focus on following 2 functions of $\mathcal{H}$ACA$pK$.

- **Construction of an H-matrix**
  - Step 1: Construct a cluster tree based on mesh
  - Step 2: Construct an H-matrix structure
  - Step 3: Fill in sub-matrices by using ACA

- **H-matrix-vector multiplication (HMVM)**
  - Highest frequency of use in application
  - Most time-consuming part in iterative linear solvers
Performance test of \( \mathcal{H} \text{ACApK} \) on serial computing

Basic performance is examined using serial computing.

- Execution times of \( \mathcal{H} \text{ACApK} \) is compared with \( \mathcal{H} \text{Lib} \).

Computer:
- Xeon X5680 12core (6core \( \times \) 2socket) 3.33GHz, 24GB
- Only 1 core is used.

The number of unknowns: \( N \)
- \( \text{case 1: } N = 1,000 \)
- \( \text{case 2: } N = 10,400 \)

Test model
Execution time when constructing H-matrices

$\mathcal{H}$ACA$pK$ is slightly faster than $\mathcal{H}$Lib.

Most time-consuming part (calculations of entries) is common to both libraries.

<table>
<thead>
<tr>
<th>$\varepsilon_{ACA}$</th>
<th>$N=1,000$</th>
<th>$N=10,400$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Hacapk</td>
<td>HLib</td>
</tr>
<tr>
<td>$1.0E\ - \ 3$</td>
<td>0.41</td>
<td>0.48</td>
</tr>
<tr>
<td>$1.0E\ - \ 4$</td>
<td>0.49</td>
<td>0.57</td>
</tr>
<tr>
<td>$1.0E\ - \ 5$</td>
<td>0.56</td>
<td>0.64</td>
</tr>
</tbody>
</table>
Execution time when performing an HMVM

Execution time of $\mathcal{H}$ACApK is half of $\mathcal{H}$lib

- Difference in data structures and languages
  - $\mathcal{H}$lib: Tree structure of C++ pointers
  - $\mathcal{H}$ACApK: Simple structure of arrays of Fortran

<table>
<thead>
<tr>
<th>$\varepsilon_{ACA}$</th>
<th>$N=1,000$</th>
<th>$N=10,400$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Hacapk</td>
<td>H-lib</td>
</tr>
<tr>
<td>$1E-3$</td>
<td>8.1e-4</td>
<td>1.7e-3</td>
</tr>
<tr>
<td>$1E-4$</td>
<td>9.4e-4</td>
<td>2.0e-3</td>
</tr>
<tr>
<td>$1E-5$</td>
<td>1.1e-3</td>
<td>2.3e-3</td>
</tr>
</tbody>
</table>

**Speed-up**

$\mathcal{H}$ACApK

$\mathcal{H}$-lib
1 Background & Objectives
2 Overview of H-matrices with ACA
3 New H-matrices code $\mathcal{HACA}_pK$
4 Parallelization of H-matrices in $\mathcal{HACA}_pK$
5 Future work
Parallelization of H-matrices in $HACApK$

- **When constructing an H-matrix**
  - Only step 3 (time-consuming part) is parallelized.
  - Any MPI communication is not needed.

  
<table>
<thead>
<tr>
<th>Step 1</th>
<th>Step 2</th>
<th>Step 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Make Cluster tree</td>
<td>Make H-matrix structure</td>
<td>Fill in sub-matrices by ACA</td>
</tr>
</tbody>
</table>

  - Redundant computation on all processes
  - Parallel computing

- **When performing an HMVM**
  - All MPI processes have the full multiplicand vector.
  - MPI communications are needed to realize it.

- **In both parallelization above**
  - Arithmetic are performed in units of sub-matrix.
  - Assignment to each process is a collection of sub-matrices.
How to assign sub-matrices to each process

Same assignment: when constructing an H-matrix when performing HMVM

Process | Assigned data
--- | ---
P0 | 1. Divide \([1, N]\) into \(N_p\) subsets.
  \[1 = l_1 < \cdots < l_k < \cdots < l_{N_p + 1} = N + 1\]
  and associate a subset with process

P1
P2
P3
P4
P5
P6
P7

2. Assign each block to a process based on the upper-left entry and \(l_k\).

3. Estimate the computational amount on each process.

4. Adjust load-balancing by moving separators \(l_k\).
For an HMVM, $y := \tilde{A}x$

Arithmetic in each process and MPI comm. is required.

- Suppose that HMVM is carried out at least several times.
- Arithmetic in each process results in part of the result of HMVM.
- In order to make full vector in all processes, each process calls SEND and RECV $(N_p - 1)$ times.
Performance test of $\mathcal{H}$ACA$p$K (Flat-MPI version)

Parallel scalability is examined
- when constructing $H$-matrices
- when performing HMVM

Computer: Fujitsu FX10 at the university of Tokyo
Processor: SPARC64™ Ixfx (16 cores/node)
Memory: 32GB
Network: 5 GB/s, Tofu.

The number of unknowns: $N$
- case 1: $N = 1,000$
- case 2: $N = 10,000$
- case 3: $N = 100,000$
Parallel Scalability of $\mathcal{H}$ACApK (Flat-MPI version)

- The larger the data size becomes, the better parallel scalability $\mathcal{H}$ACApK attains in both cases.
- Better parallel scalability is shown when constructing H-matrices.
- Parallel speed-up in a HMVM strongly depends on the data size.
Parallel Scalability of $\mathcal{H}$ACA$p$K (Flat-MPI version)

- The larger the data size becomes, the better parallel scalability $\mathcal{H}$ACA$p$K attains in both cases.
- Better parallel scalability is shown when constructing H-matrices.
- Parallel speed-up in a HMVM strongly depends on the data size.

![Graphs showing speed-up vs. number of MPI processes for constructing H-matrices and H-matrix vector multiplication.]
To improve the parallel scalability in HMVM

We have developed a hybrid MPI+OpenMP version.

- The speed-up limit is caused by the MPI communication costs because the load-imbalance isn’t significant in the construction.

- In hybrid version, arithmetic below is needed instead of comm.

Add all using the atomic option of OpenMP

- Conflict when adding into same place
Effects of using Hybrid MPI+OpenMP in HMVM (FX10)

We examined speedup vs. the time of the Flat-MPI ver. on 1 node.

- Parallel scalability is improved in the case of hybrid MPI+OpenMP by reducing MPI communication cost.
- Speed-up reaches a limit around 96-cores in case of Flat-MPI.

Parallel scalability when performing an H-matrix vector multiplication
Summary

- Development of a new parallel $\mathcal{H}$-matrices code $\mathcal{H}$ACApK
- Static electric field analyses up to about 30 million unknowns.
- Performance evaluation of $\mathcal{H}$ACApK
  - On serial computing
    - $\mathcal{H}$ACApK was not slower than the existing software $\mathcal{H}$-lib.
  - Flat-MPI version on a SMP cluster system (Fujitsu FX10)
    - Saturation of speedup was observed in HMVM.
    - Better parallel scalability was shown when constructing the H-matrix.
  - Hybrid MPI+OpenMP version
    - The parallel scalability was improved from the flat-MPI version.
    - The best result was observed when 8 OpenMP threads are used in each SMP node.
Future work

- Performance evaluation of $\mathcal{H}$ACApK
  - using earthquake cycle simulation
  - using quantum mechanics

- Improvement of functions for HMVM
  - Reducing communications among MPI processes
  - Overlapping of communication and computation
  - Avoiding conflicts among threads
  - Application of auto-tuning

- Others
  - Parallelization of full H-matrices construction
  - Development of efficient linear solver by using H-matrices
  - Enhancement of functions
  - etc.